

Chapter 4 A Meta-Model for Specifications

The previous chapter proposed an architecture for automating the generation of concurrent designs for real-time software. The proposed architecture identified three meta-models for design information: a meta-model for specifications, a meta-model for concurrent designs, and a meta-model for target environment descriptions. This chapter describes in detail the specification meta-model.

To provide an initial structuring for real-time problems, analysts often build a behavioral model based upon data/control flow diagrams, augmented with state-transition diagrams (sometimes referred to as finite automata or finite-state machines). One widely used method to create and describe such a behavioral model is known as Real-Time Structured Analysis, or RTSA. When applied to a real-time problem, RTSA produces a set of hierarchical data/control flow diagrams, with supporting state-transition diagrams to depict the operations within control transformations, with pseudo-code specifications to outline the operations within data transformations, and with a data dictionary to detail the data flows represented in the flow diagrams.

After creating such a behavioral model, designers then consider how to map elements from the data/control flow diagrams onto a structure that can lead to an effective and efficient software implementation. Sometimes a designer intends to produce a

sequential design, at other times a concurrent design. The research described in this dissertation, assumes that the designer requires a concurrent design, that is, a design composed of tasks and modules.

In order to automate the reasoning used by designers to transform data/control flow diagrams into concurrent designs, the knowledge available to designers must be represented in a machine-processible form. A substantial component of the needed knowledge involves representing and reasoning about specifications represented as flow diagrams. Subsequent sections of the current chapter define this knowledge as a semantic meta-model for specifications, based on RTSA data/control flow diagrams and other supporting information. While real-time problems can be modeled with RTSA notation, RTSA specifications can be modeled with the specification meta-model defined in this chapter. The specification meta-model defines each element that composes a RTSA specification as a semantic concept and also places those concepts in semantic relation to one another; thus, the specification meta-model can be called a semantic meta-model.

Using the semantic meta-model, as defined below, RTSA specifications can be represented and reasoned about. One form of reasoning supported by the meta-model enables a RTSA specification to be examined for the presence of semantic concepts. This form of reasoning allows the data and control transformations, the terminators, and the data and event flows in a RTSA data/control flow diagram to be classified more specifically as semantic concepts from the meta-model. For example, a data transformation on the edge of a data/control flow diagram might be classified as an

Asynchronous Device Input Object. The most specific concepts in the semantic meta-model, called leaf concepts, appear as leaf nodes in a concept hierarchy. A second form of reasoning supported by the semantic meta-model determines whether or not a RTSA specification consists entirely of leaf concepts. For example, a data transformation classified as an Asynchronous Device Input Object, having no subordinate concepts in the semantic meta-model, would be recognized as a leaf concept, while a data transformation classified as an Asynchronous Device Interface Object, having several subordinate concepts in the semantic meta-model, would be recognized as a non-leaf concept. A third form of reasoning facilitated by the semantic meta-model allows any component in a RTSA specification, after having been classified as a concept in the semantic meta-model, to be evaluated against a concept definition to determine whether or not the component represents a valid instance of the concept. For example, if the semantic meta-model requires an Asynchronous Device Interface Object to be the recipient of an interrupt from a device, then any component of a RTSA specification that depicts a valid instance of the concept Asynchronous Device Interface Object must meet this requirement.

4.1 The Concept Hierarchy

Real-Time Structured Analysis, or RTSA, provides a small number of symbols, shown in Figure 5, for documenting data/control flow diagrams.¹ While flexible and useful for expressing the relationships between data flows and the processes that

¹The semantic meta-model defined in this dissertation omits the RTSA symbol for continuous data flow. Continuous data flows do not play a large part in the specification of real-time systems for digital computers.

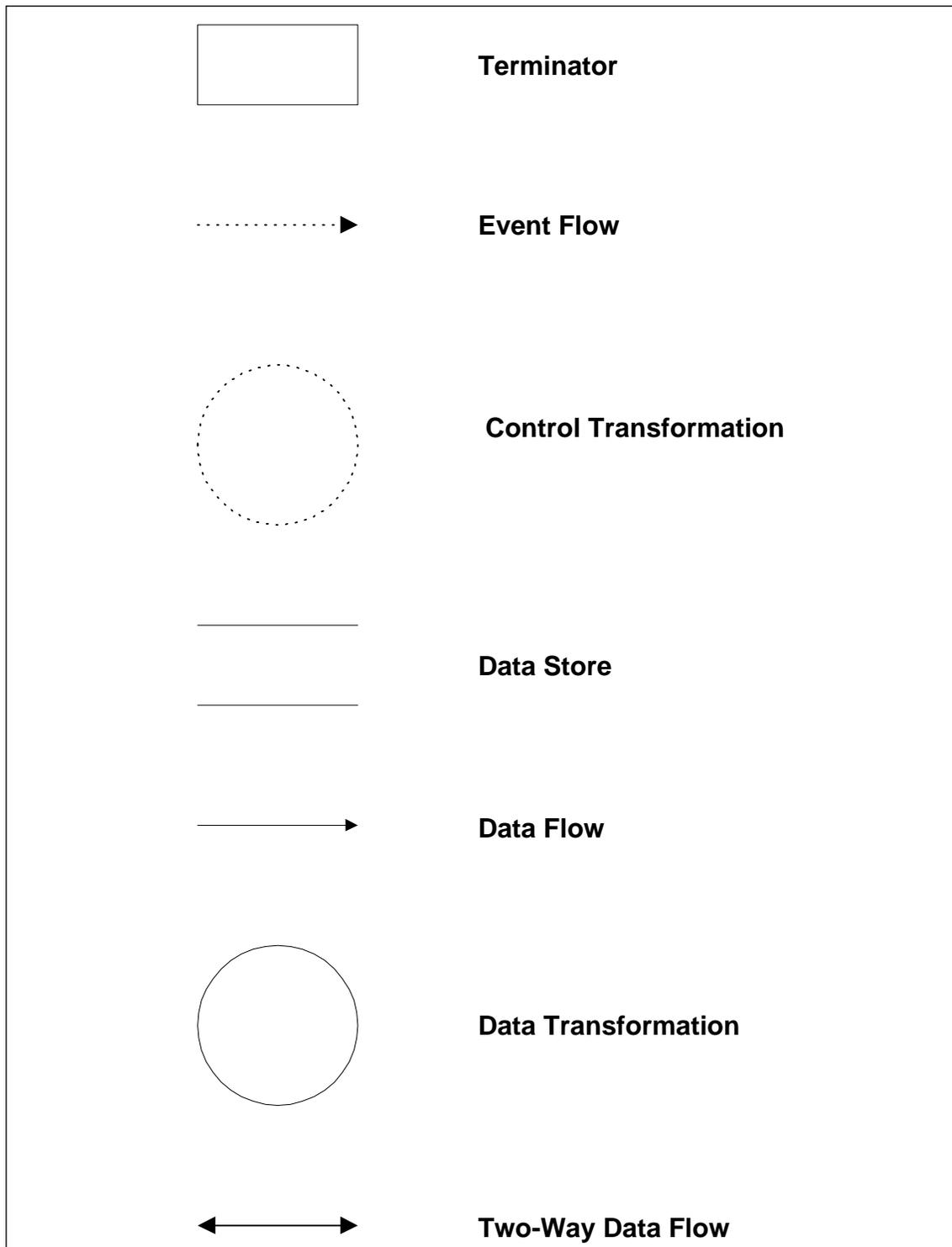


Figure 5. Syntactic Elements for Composing RTSA Data/Control Flow Diagrams

transform data, and the relationships between event flows and control processes and the processes that they control, the symbols alone do not provide the richness of semantic content needed to reason about the generation of concurrent designs. For this reason, Gomaa, in his Concurrent Object-Based Real-time Analysis method, or COBRA, restricts the combinations of RTSA symbols used to model real-time problems, and then attributes semantic significance to the allowed combinations. The semantic meta-model for specifications, as defined in this chapter, begins with the ideas proposed in COBRA, and then provides a sufficiently rigorous definition of the allowed semantic concepts so that they might serve as a basis for automating the analysis of data/control flow diagrams and the generation of concurrent designs.

A major aspect of the specification meta-model includes a concept hierarchy, where each child concept specializes, using an **is-a** relationship, its parent concept. A relatively simple notation represents the concept hierarchy. A circle denotes each semantic concept within the hierarchy. A directed arc, drawn from a child concept to the parent concept(s), represents each **is-a** relationship.² Being rather complex, the complete concept hierarchy requires several, separate diagrams. Whenever a concept appears on a diagram as two concentric circles then another diagram exists that continues the hierarchy from that concept.

²The circles and arcs in the specification meta-model represent semantic concepts and inheritance, respectively. The reader should take care not to confuse the notation used to describe the specification meta-model with the notation used by RTSA to represent data/control flow diagrams. The two notations are unrelated.

The definition of the semantic meta-model for specifications begins with a first-level division of concepts into Specification Element and Specification Addenda, as shown in Figure 6. Specializations of the concept Specification Element represent components of data/control flow diagrams, while specializations of the concept Specification Addenda represent additional information that, while not depicted on data/control flow diagrams, is needed to make design decisions. Each of these first-level divisions is described further below.

4.1.1 Specification Elements

The concept Specification Element generalizes the concepts that compose data/control flow diagrams. Specializations of the concept Specification Element lead, through some intermediate concepts, first to the syntactic elements of a RTSA

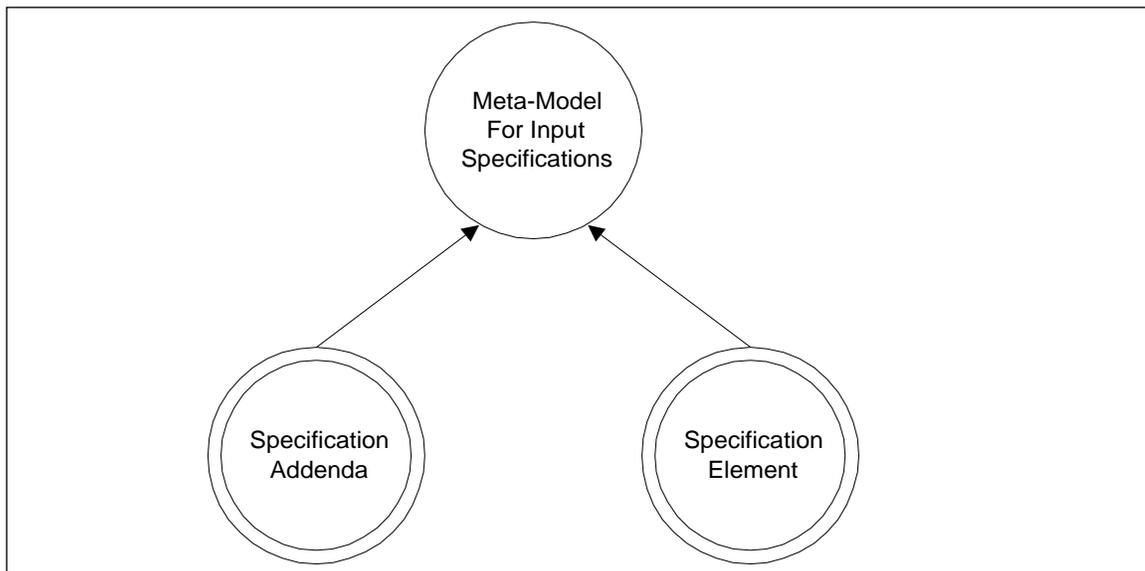


Figure 6. First-Level Division of Concepts

data/control flow diagram, and then, with increasing specialization, to the leaf concepts of the semantic meta-model. This classification scheme allows the presence of semantic concepts in a given input specification to be inferred from the syntactic elements of RTSA data/control flow diagrams.

4.1.1.1 RTSA Syntactic Element Classification

Figure 7 depicts the specialization of Specification Element to the point where RTSA syntactic elements (refer back to Figure 5) can be denoted. The first specialization of Specification Element defines Node, Directed Arc, and Two-Way Arc. These abstract concepts cannot be represented directly within an input specification, but must be inherited by more specialized concepts. Informal definitions for each of these abstract concepts follow.

The concept Node is defined to be a point within a grid. A Node must be named, and every Node within a given grid must have a unique name. No Node can be named "System", as this name is reserved as a source for timer event flows. A Node is also repeatable; so, any given, uniquely-named Node in a grid can have more than one instance.

The abstract concept Directed Arc is defined to be a directed link between two Nodes in a grid. Each Directed Arc denotes a link from one Node (called the source) to a second Node (called the sink) in a grid. A Directed Arc can be labeled with a name. No Directed Arc can have the same Node as both the source and the sink. The abstract concept Two-Way Arc is defined to be a directionless link between two Nodes in a grid.

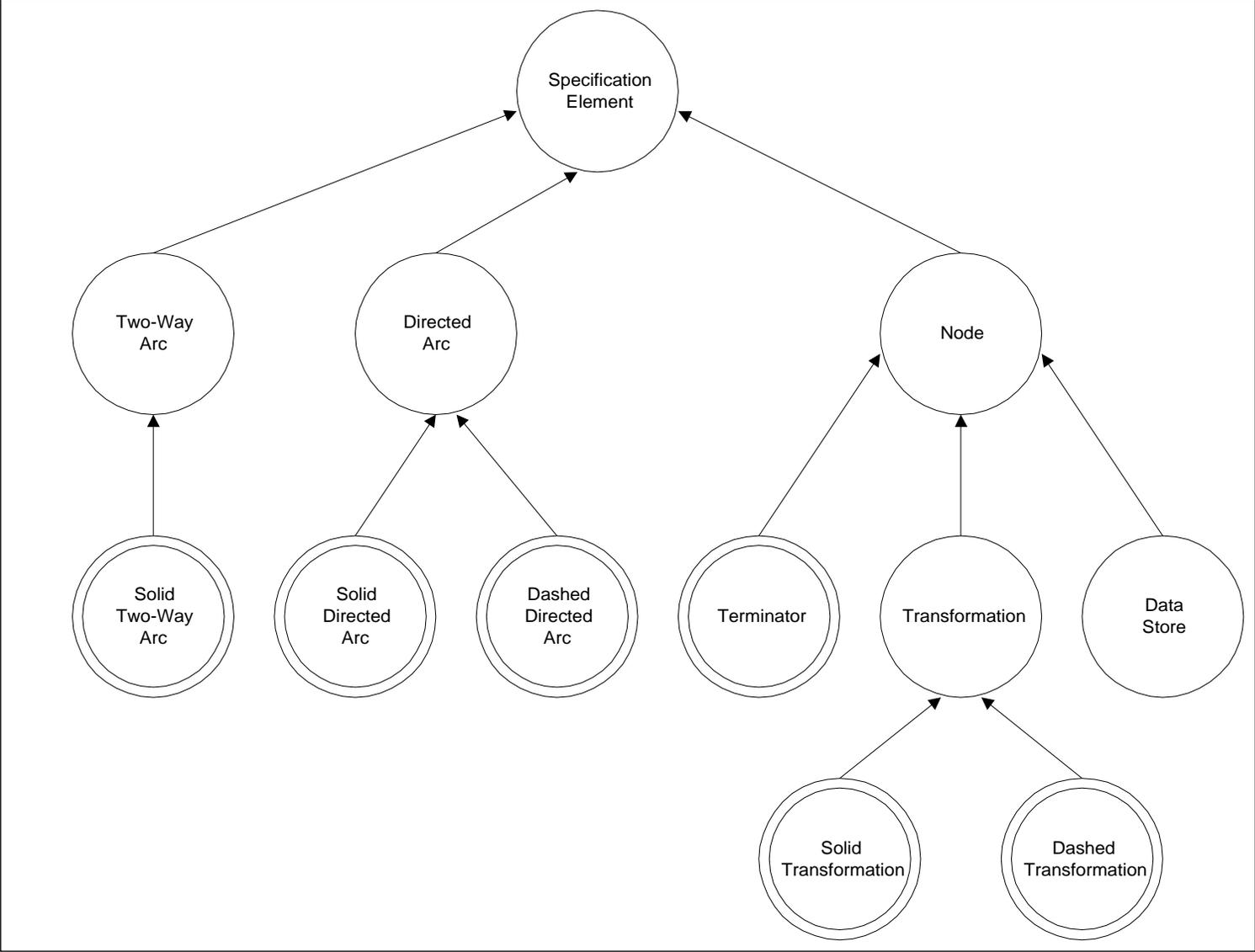


Figure 7. Specialization of Specification Element to RTSA Syntactic Elements

Each Two-Way Arc denotes a link between one Node (called the right) and a second Node (called the left) in a grid. A Two-Way Arc can be labeled with a name. No Two-Way Arc can have the same Node as both the left and the right.

The semantic meta-model provides three specializations of the concept Node, as shown in Figure 7. The concepts Terminator and Data Store can be used to represent, in an input specification, the RTSA syntactic elements of the same name. The abstract concept Transformation (sometimes referred to, for brevity, as Transform) further specializes the concept Node, but cannot be represented directly in an input specification. The concept Transformation is a Node that can be numbered. Transformation numbers can take a hierarchical form, where each Transformation number is represented as a series of one or more natural numbers. When a Transformation number contains more than one natural number, each natural number is separated from the one following it by a period. For example, valid Transformation numbers include: "1.0", "1.1.2.3", "1", "7.2.3", and so on. Within a set of Transformations in the same input specification no two Transformation numbers can be identical.

The abstract concept Transformation is specialized into two concepts that can be represented directly using RTSA syntactic elements. The concept Solid Transformation represents the RTSA syntactic element called Data Transformation, while the concept Dashed Transformation represents the RTSA syntactic element called Control Transformation.

The semantic meta-model provides two specializations of the concept Directed Arc, as illustrated in Figure 7. The concepts Solid Directed Arc and Dashed Directed Arc represent the RTSA syntactic elements called Data Flow and Event Flow, respectively. The semantic meta-model includes only one specialization of the concept Two-Way Arc. This specialization, shown as Solid Two-Way Arc in Figure 7, represents the RTSA syntactic element called Two-Way Data Flow.

Using the concept hierarchy for Specification Element, as explained to this point, RTSA data/control flow diagrams can be represented. Further specialization of the concept hierarchy allows additional semantic concepts to be identified and denoted.

4.1.1.2 Semantic Element Classification

COBRA, restricts the combinations of RTSA symbols used to model real-time problems, and then attributes semantic significance to the allowed combinations. In the manner suggested by COBRA, the semantic meta-model further classifies the syntactic elements of RTSA in order to enable additional semantic concepts to be recognized and represented.

4.1.1.2.1 Terminators

The specification meta-model allows three types of Terminator to be distinguished, as shown in Figure 8. The concept Device is a Terminator that corresponds to hardware in the problem domain. The concept External Subsystem is a Terminator that denotes an external, independent, subsystem that communicates with the subsystem represented by a given input specification. The concept User Role is a

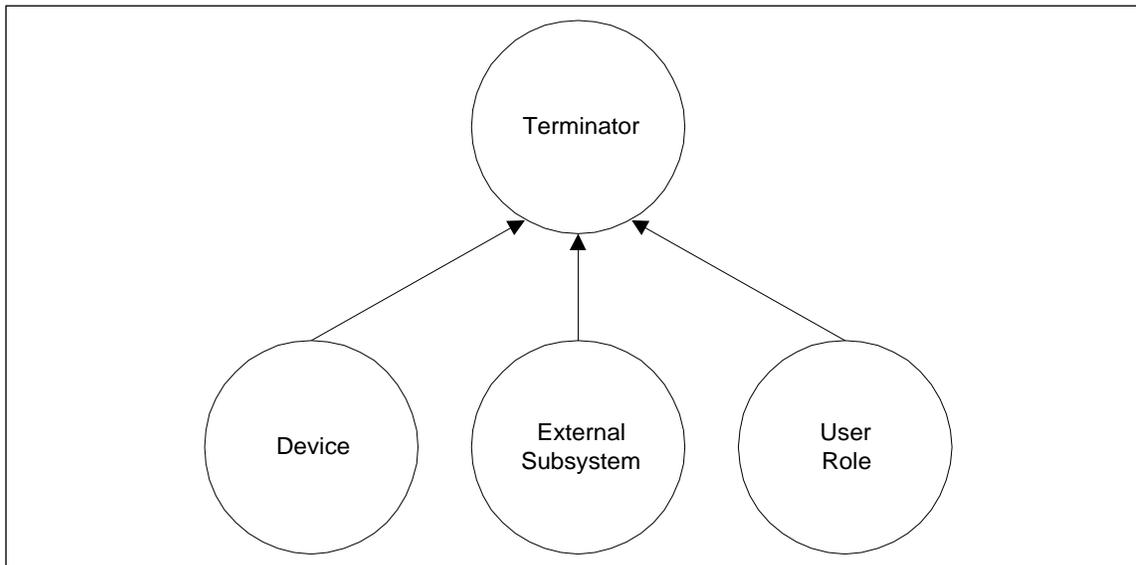


Figure 8. The Terminator Classification Hierarchy

Terminator that represents a human operator who interacts with the system using a dialog of commands and responses. These three concepts, Device, External Subsystem, and User Role, appear as leaf concepts in the semantic meta-model.

4.1.1.2.2 Solid Transformations

As shown in Figure 9, the semantic meta-model distinguishes among a variety of uses for the concept Solid Transformation. Fundamentally, two types of Solid Transformations are distinguished. The concept Interface Object denotes a Solid Transformation that exchanges events or data with a Terminator. The concept Function denotes a Solid Transformation that does not exchange events or data with a Terminator. Each Interface Object can exchange data with only one Terminator. The concept Interface Object can be further specialized through the concepts User-Role Interface Object (an Interface Object that exchanges data with a User-Role), Subsystem Interface

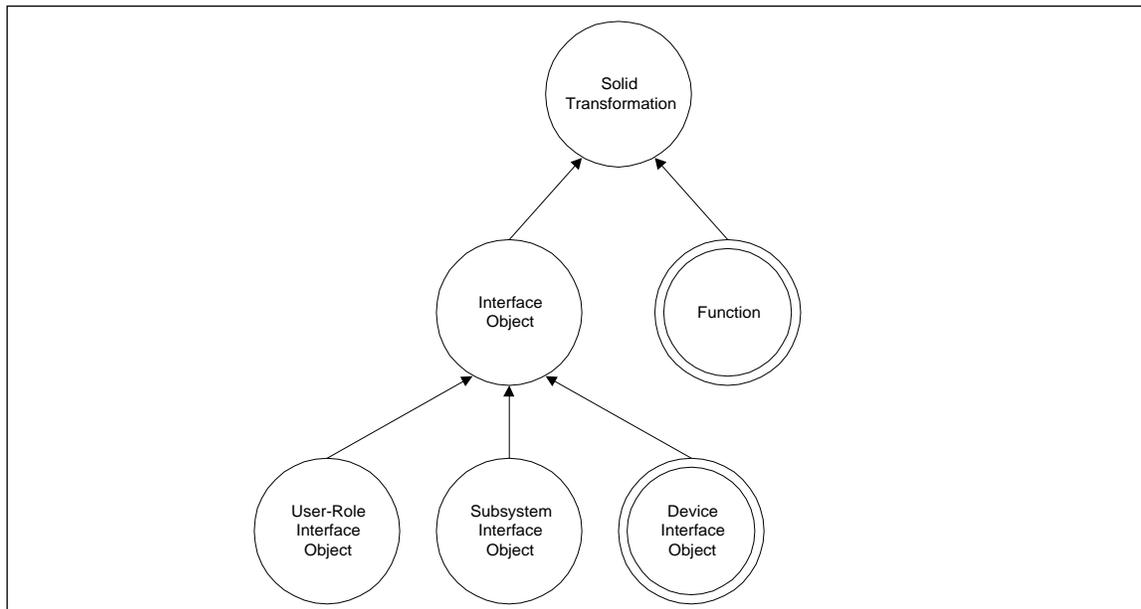


Figure 9. The Solid Transformation Classification Hierarchy

Object (an Interface Object that exchanges data with an External Subsystem), and Device Interface Object (an Interface Object that exchanges data with a Device). The concepts User-Role Interface Object and Subsystem Interface Object appear as leaf concepts in the semantic meta-model; however, as shown in Figure 9, the concepts Device Interface Object and Function can be further specialized.

4.1.1.2.3 Device Interface Objects

The semantic meta-model enables Device Interface Objects to be specialized based on the characteristics, or requirements, of the device for which they provide an interface. In fact, the various specializations of Device Interface Object included in the specification meta-model provide a rich set of concepts, as shown in Figure 10. Device

Interface Objects can be classified based on the type of data exchange with a Device. A Device Interface Object that only receives data from an associated Device can be specialized as a Device Input Object. A Device Interface Object that only sends data to an associated Device can be specialized as a Device Output Object. A Device I/O Object both receives data from and sends data to an associated Device.

Device Interface Objects can also be classified based on the impetus for data exchanges with an associated Device. A Device Interface Object that must periodically tend to an associated Device can be specialized as a Periodic Device Interface Object. A Device Interface Object that waits for an associated Device to request attention can be specialized as an Asynchronous Device Interface Object. A Device Interface Object that tends to an associated Device only when requested to do so by a Function can be specialized as a Passive Device Interface Object.

The specializations of Device Interface Object based on the direction of data exchange and on the impetus for data exchange can be combined, using multiple inheritance, to form nine varieties of Device Interface Object, as shown in Figure 10. These nine concepts appear as leaf concepts within the semantic meta-model for specifications.

4.1.1.2.4 Control Objects

Before completing the discussion of the Solid Transformation classification hierarchy by defining the specializations of Function, the concept Control Object must be introduced, followed by the specializations of Solid Directed Arc and Dashed Directed

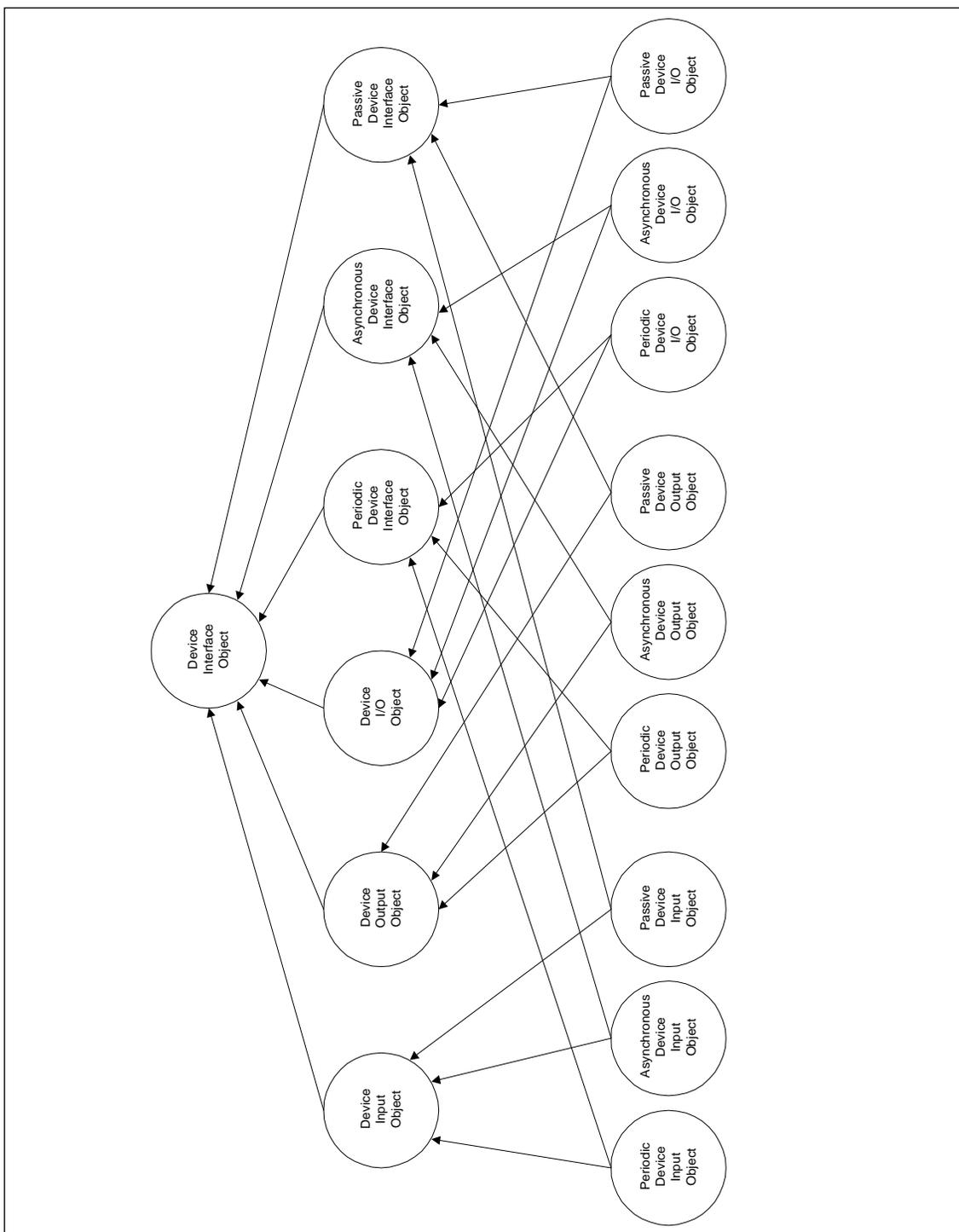


Figure 10. The Device Interface Object Classification Hierarchy

Arc. A Control Object, the sole specialization of Dashed Transformation supported currently by the semantic meta-model, denotes a Transformation that contains a related state-transition diagram. Only Dashed Directed Arcs may enter or leave a Control Object. Each incoming Dashed Directed Arc represents an event that the state-transition diagram should recognize, while each outgoing Directed Arc represents some control action, initiated by a transition contained within the state-transition diagram. For completeness, Figure 11 shows the specialization of a Dashed Transformation as a Control Object. Control Object appears as a leaf concept in the semantic meta-model.

4.1.1.2.5 Solid Directed Arcs

The semantic meta-model allows the concept Solid Directed Arc to be specialized for three separate uses, as shown in Figure 12. One specialization, Internal Data Flow, denotes the flow of data between Solid Transformations. A second specialization, External Data Flow, denotes the exchange of data between an Interface Object and a Terminator. The third specialization, Data-Store Data Flow, represents data exchange between a Solid Transformation and a Data Store.

Further specializations of the concept Internal Data Flow include Stimulus and Response. The concept Stimulus represents most data that flows between Solid Transformations. The concept Response denotes the special case where data is sent from one Solid Transformation, say A, to another Solid Transform, say B, in reply to a Stimulus that was received by A from B.

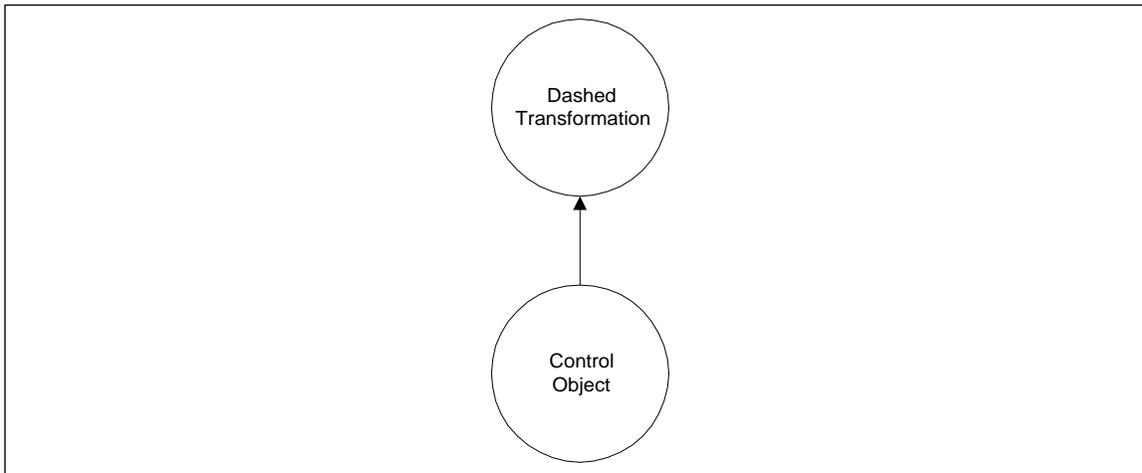


Figure 11. The Dashed Transformation Classification Hierarchy

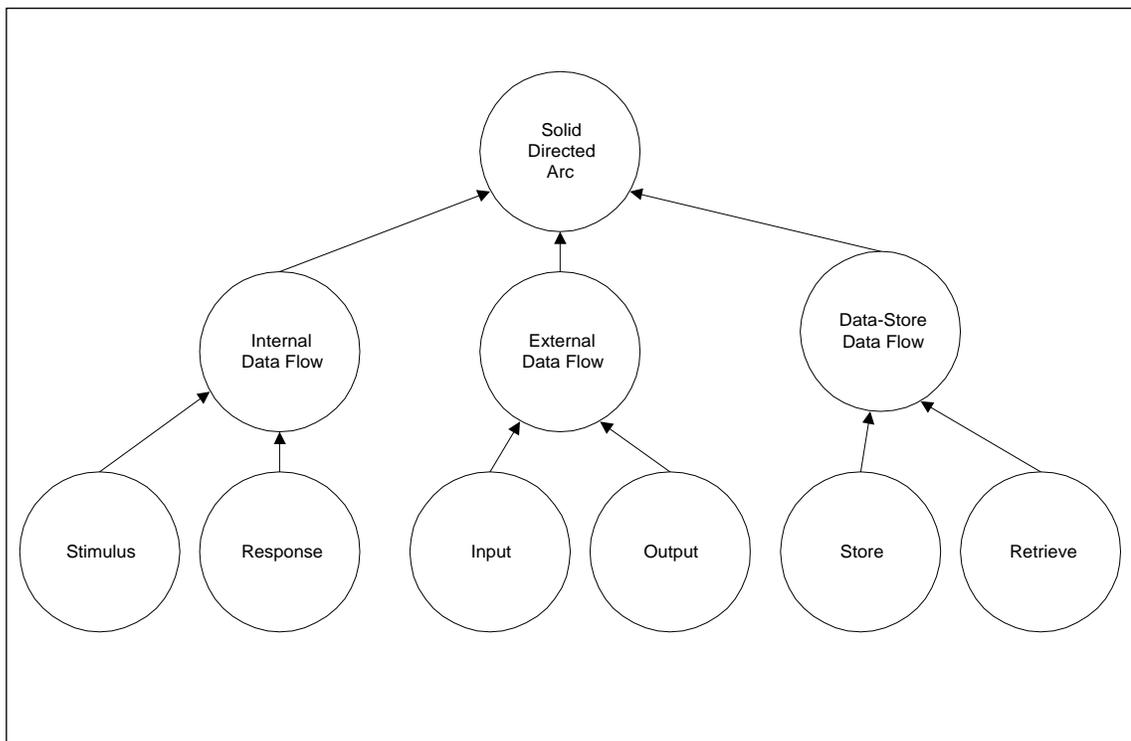


Figure 12. The Solid Directed Arc Classification Hierarchy

Further specializations of the concept External Data Flow include Input and Output. The concept Input denotes data that flows from a Terminator to an Interface Object. The concept Input includes an attribute, maximum-rate, that can contain the maximum number of inputs per second generated by an associated Terminator. The concept Output denotes data that flows from an Interface Object to a Terminator.

Further specializations of the concept Data-Store Data Flow include Store and Retrieve. The concept Store denotes data that flows from a Solid Transform to a Data Store, while the concept Retrieve denotes data that flows from a Data Store to a Solid Transform. Stimulus, Response, Input, Output, Store, and Retrieve appear as leaf concepts within the semantic meta-model.

4.1.1.2.6 Dashed Directed Arcs

The semantic meta-model specializes the concept Dashed Directed Arc as the concept Event Flow. This specialization, while not strictly required, allows the meta-model to be expanded should additional uses be found for the Dashed Directed Arc. The concept Event Flow can be specialized, as shown in Figure 13, into two abstract concepts: Normally-Named Event Flow and Specially-Named Event Flow. The abstract concept Specially-Named Event Flow denotes an event flow that is labeled with a name from among a set of reserved event-flow names. At the present time, the set of reserved event-flow names includes: Disable, Enable, and Trigger. The abstract concept Normally-Named Event Flow denotes an event flow that is labeled with a name that is not among the set of reserved event-flow names.

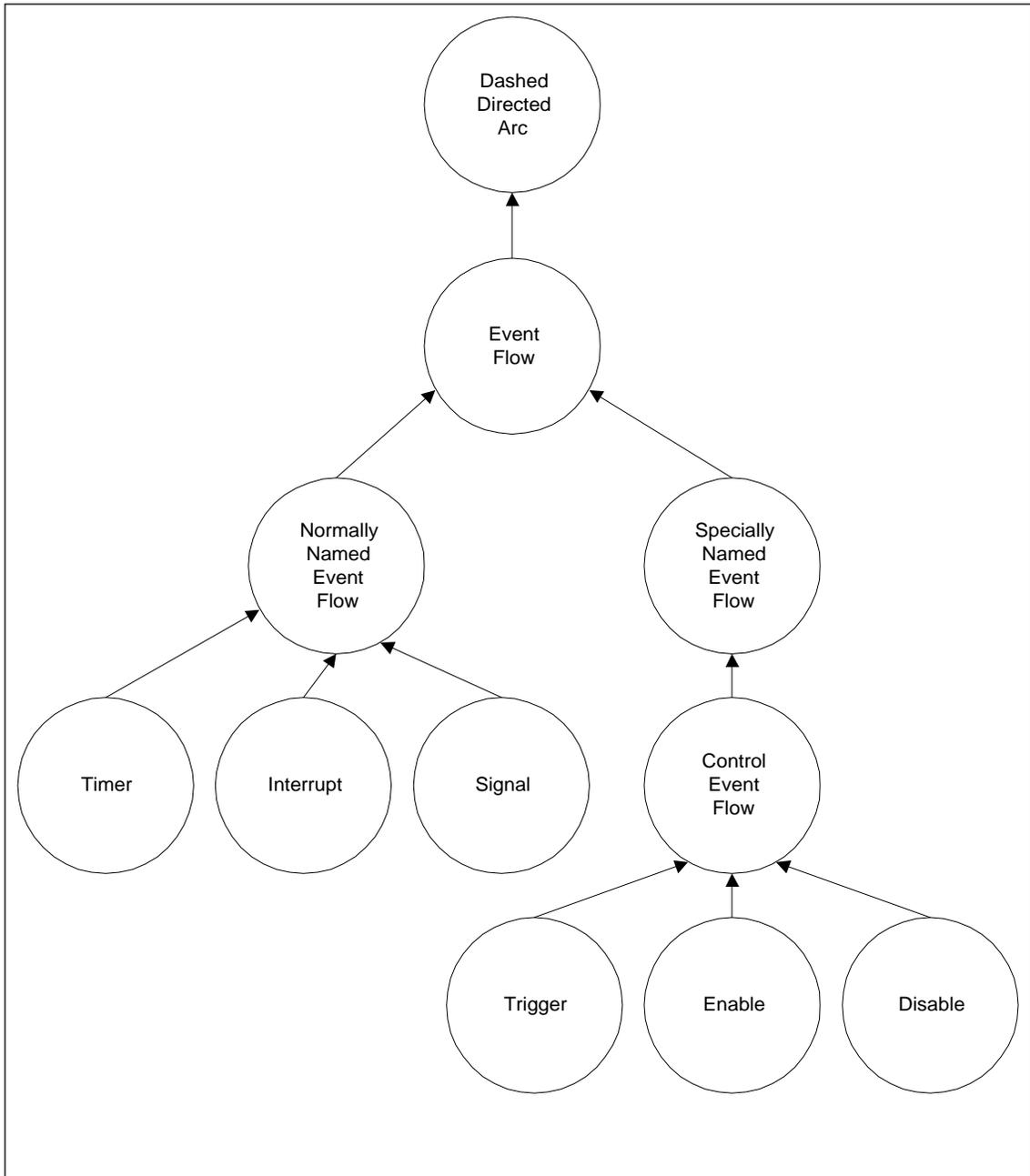


Figure 13. The Dashed Directed Arc Classification Hierarchy

The abstract concept Specially-Named Event Flow can be specialized as a Control Event Flow. The concept Control Event Flow denotes a Specially-Named Event Flow that originates from a Control Object (the source) and flows to a Solid Transformation (the sink). A Control Event Flow must be labeled from a partition of the names reserved for a Specially-Named Event Flow. At the present time, the valid partition includes the entire set of reserved event flow names (that is, Disable, Enable, and Trigger).

The concept Control Event Flow can be further specialized into three concepts: Trigger, Enable, or Disable. The label of each of these must correspond to the concept name; for example, any instance of the concept Trigger is labeled with the name Trigger. Whenever a Trigger flows between a Control Object and a Solid Transformation, then no other Control Event Flow may flow between the same Control Object and Solid Transformation. Whenever an Enable flows between a Control Object and a Solid Transformation, then a Disable must also flow between the same Control Object and Solid Transformation; however, no other Control Event Flow may flow between the same Control Object and Solid Transformation. Disable, Enable, and Trigger appear as leaf concepts in the semantic meta-model.

The abstract concept Normally-Named Event Flow can be specialized as a Timer, an Interrupt, or a Signal. The concept Timer denotes a signal that is received periodically from the underlying operating system or hardware clock. The concept Timer includes an attribute, period, that denotes the frequency in seconds with which a specific instantiation

of Timer will be generated. The value contained by the attribute period must exceed zero. The source of any Timer must be "System".

The concept Interrupt denotes a signal that is received by an Asynchronous Device Interface Object from a Device. The concept Interrupt includes an attribute, maximum-rate, that can contain the maximum number of interrupts per second generated by an associated Device. The attribute maximum-rate is used only in situations where a Device generates interrupts without any associated data (i.e., Input). In other cases, the maximum-rate for the input data suffices.

The concept Signal denotes an event sent between two Transformations within a data/control flow diagram. No data is associated with a Signal. Timer, Interrupt, and Signal appear as leaf concepts in the semantic meta-model.

4.1.1.2.7 Functions

The Function classification hierarchy is shown in Figure 14. The concept Function can be specialized into two concepts. Definitions for these concepts are given below. The concept State-Independent Function denotes a Function that is not the sink for a Control Event Flow. The concept State-Dependent Function denotes a Function that is the sink for a Control Event Flow.

The concept State-Independent Function can be further classified as a Periodic Function or an Aperiodic Function. The concept Periodic Function represents a State-Independent Function that is invoked periodically. A Periodic Function must be the sink for one, and only one, Timer. The concept of Aperiodic Function denotes a

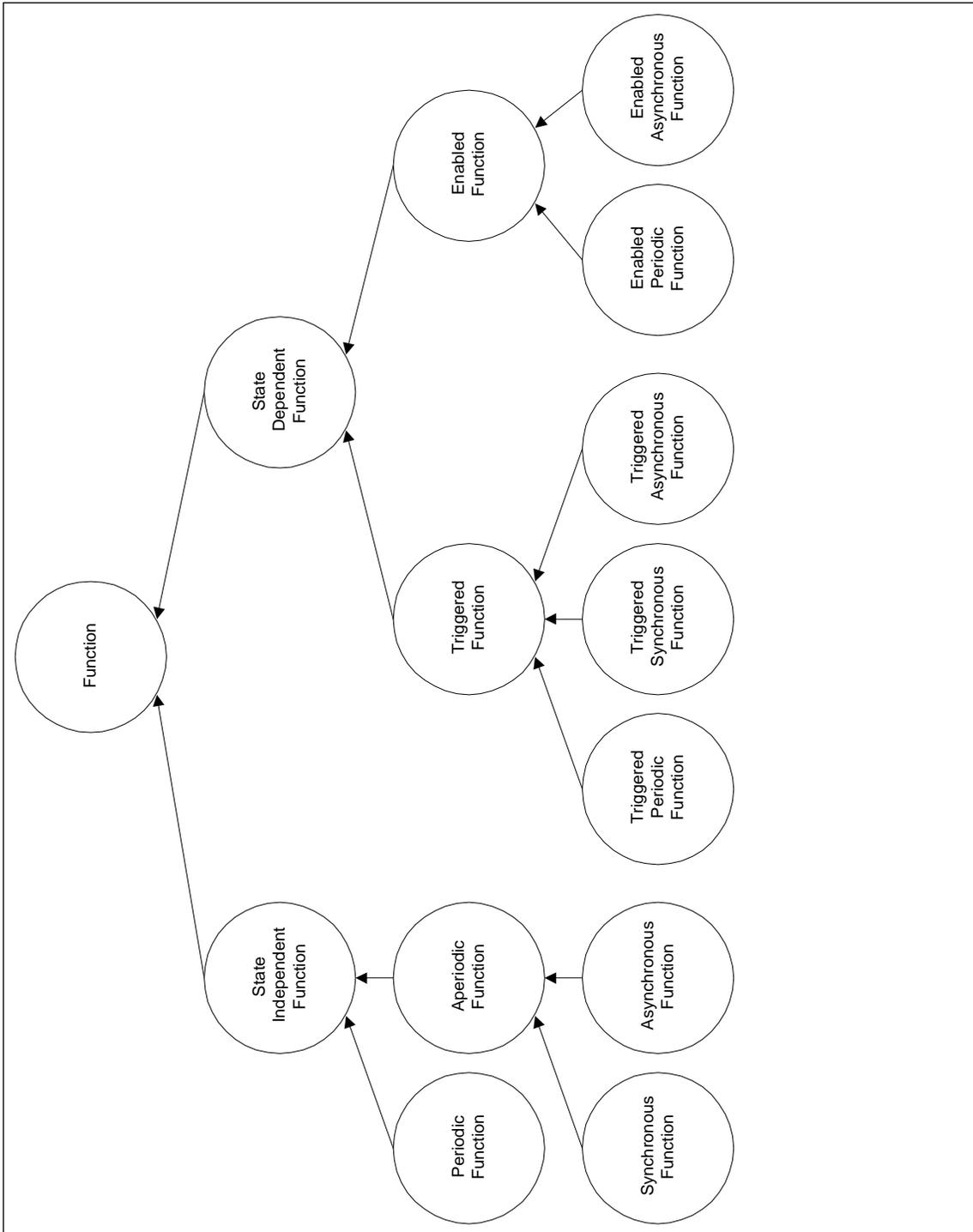


Figure 14. The Function Classification Hierarchy

State-Independent Function that is invoked on demand. An Aperiodic Function cannot be the sink for a Timer. The concept Aperiodic Function can be further specialized as either an Asynchronous Function or a Synchronous Function. The concept Synchronous Function denotes a function that does not operate independently, but instead operates synchronously with some other function because either: 1) the other function must wait for the results of the Synchronous Function before continuing or 2) the Synchronous Function performs its processing so quickly that no advantage is gained from letting the function operate independently. The concept Asynchronous Function denotes a function that operates independently. The concepts Periodic Function, Synchronous Function, and Asynchronous Function appear as leaf concepts in the semantic meta-model.

The concept State-Dependent Function can be further classified as a Triggered Function or an Enabled Function. The concept Triggered Function denotes a function that is the sink for a Trigger or for one or more Signals that originate from a Control Object. The concept Enabled Function denotes a function that is the sink for an Enable from a Control Object.

The concept Triggered Periodic Function denotes a function that, once triggered by a Control Object, executes periodically until finished. A Triggered Periodic Function must be the sink for one, and only one, Timer. The concept Triggered Synchronous Function denotes a function that completes execution during the state-transition within which the function was triggered by a Control Object. The concept Triggered Asynchronous Function denotes a function whose completion is independent of the

state-transition within which the function was triggered by a Control Object. A Triggered Asynchronous Function executes until it completes.

The concept Enabled Periodic Function denotes a function that, once enabled by a Control Object, executes periodically until disabled by a Control Object. An Enabled Periodic Function must be the sink for one, and only one, Timer. The concept Enabled Asynchronous Function denotes a function that, once enabled by a Control Object, executes independently, until disabled by the enabling Control Object. The concepts Triggered Periodic Function, Triggered Synchronous Function, Triggered Asynchronous Function, Enabled Periodic Function, and Enabled Asynchronous Function appear as leaf concepts within the semantic meta-model.

4.1.1.2.8 Solid Two-Way Arcs

The Solid Two-Way Arc classification hierarchy, shown in Figure 15, completes the specialization of Specification Element. The sole specialization of the concept Solid Two-Way Arc is Update. The concept Update denotes a bi-directional link where a Solid Transformation is on the left and a Data Store on the right of the link. The semantic sense of Update is that some information flows from the Data Store to the Solid Transformation and then flows back again in an altered form.

4.1.2 Specification Addenda

The concept Specification Addenda denotes information needed to make design decisions, but information that cannot be represented as a Specification Element. Figure

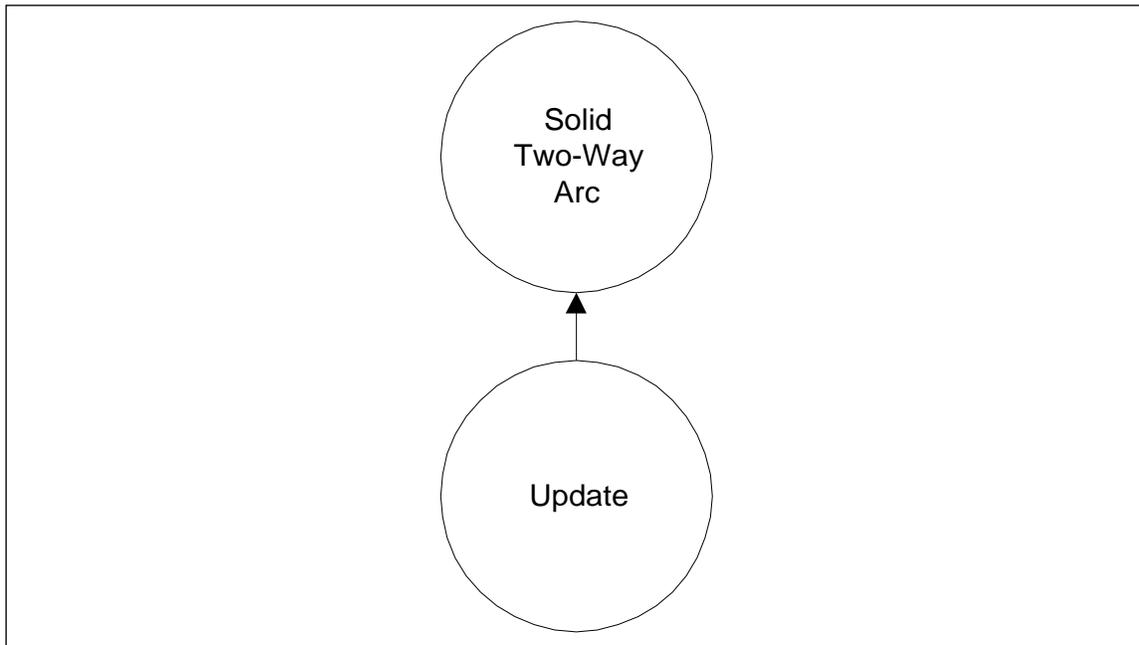


Figure 15. The Solid Two-Way Arc Classification Hierarchy

16 shows the Specification Addenda classification hierarchy defined for the semantic meta-model. Three types of Specification Addenda are defined.

4.1.2.1 Aggregation Groups

The concept Aggregation Group is used to indicate objects in a data/control flow diagram that, taken together, compose a larger physical or logical object within an application. This concept allows such relationships, not available within the notation for RTSA data/control diagrams, to be represented. Each Aggregation Group, as defined in the context of this dissertation, can consist of a Control Object and a list of the devices controlled by that Control Object. No transformation from a data/control flow diagram may be a member of more than one Aggregation Group. Each member of an Aggregation

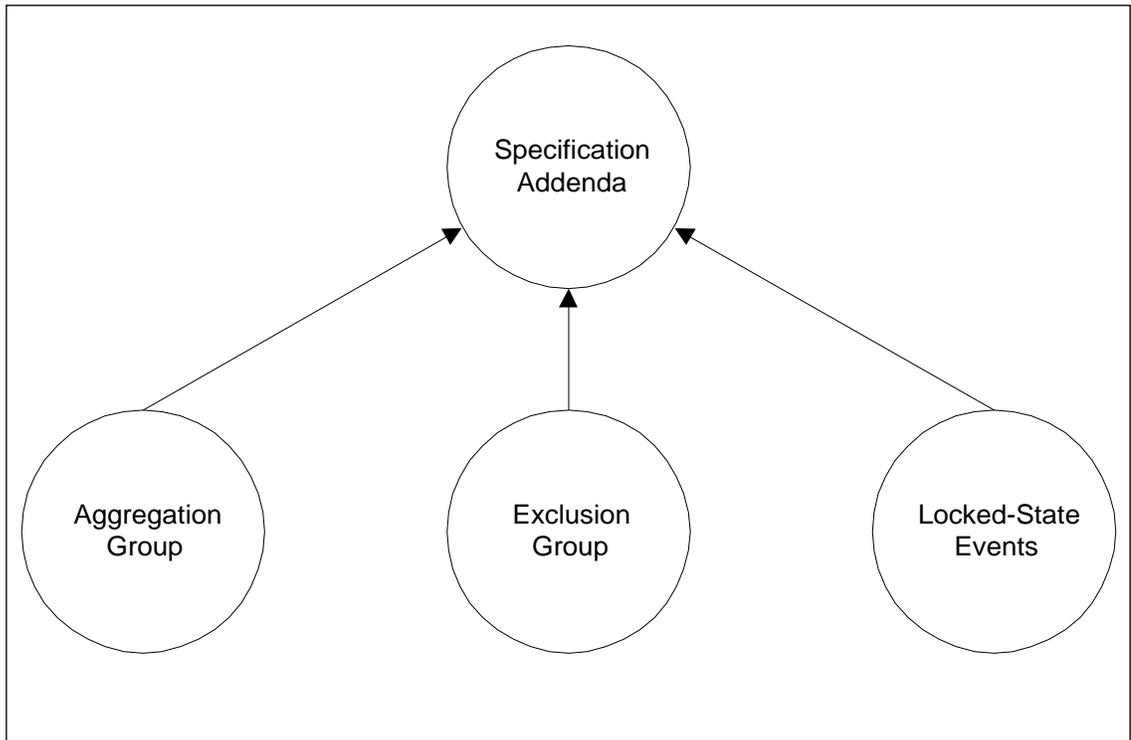


Figure 16. The Specification Addenda Classification Hierarchy

Group must correspond to the name of a Control Object or Device that exists within the data/control flow diagram to which the Specification Addenda apply. For example, an elevator control application might include an aggregate object, denoting a physical elevator, that is composed of a Control Object, Elevator Controller, and several Devices: Elevator Motor, Elevator Door, Floor Arrival Sensors, Elevator Lamps, and Elevator Buttons.

4.1.2.2 Exclusion Groups

The concept Exclusion Group relates transformations on a data/control flow diagram, where transformations within the same Exclusion Group cannot execute at the

same time. This Specification Addenda allows information regarding mutual exclusion to be represented. Such information, helpful when making certain design decisions, cannot be depicted with RTSA data/control flow diagrams. Two forms of Exclusion Group are supported. One form can relate an instance of Control Object to instances of Enabled Function that cannot execute concurrently because of ordering restrictions imposed by the Control Object. For example, an automobile cruise control system might operate in any one of three modes, increasing speed (represented by an Enabled Function, Increase Speed), maintaining speed (represented by an Enabled Function, Maintain Speed), or resuming cruise speed (represented by an Enabled Function, Resume Cruising), at any given time, but might also be prevented by control logic (represented as a Control Object, Cruise Controller) from operating in any two modes simultaneously. The second form of Exclusion Group can relate instances of periodic and asynchronous functions that cannot execute concurrently because of ordering restrictions inherent in the application. For example, a stop-and-wait protocol, where only one message can be in transmission at any point in time, might consist of an asynchronous function, Send Next Message, and a periodic function, Resend Old Message. After understanding the description of the stop-and-wait protocol, a designer might conclude that the two transformations, Send Next Message and Resend Old Message, cannot both be active simultaneously.

For each instance of Exclusion Group that refers to a Control Object, the Control Object must exist within the specification and the Exclusion Group must include at least

two instances of Enabled Function that receive an Enable from the Control Object. For Exclusion Groups that do not include a Control Object, each Exclusion Group must include at least two members. Only periodic and asynchronous functions can be included in Exclusion Groups without a Control Object. No function can be included in more than one Exclusion Group

4.1.2.3 Locked-State Events

The concept Locked-State Events relates an instance of Control Object to a list of special instances of Signal received by the Control Object. For every state within a given Control Object such that the Control Object is locked in that state until a Signal is received from one, and only one, specific Transformation, and provided that a Signal received from that specific Transformation arrives only while the Control Object is locked in a state awaiting that Signal, then the Signal can be included in the list of Locked-State Events for that Control Object. For example, consider a robot control application in which a Control Object, Robot Command Controller, signals a Transformation, Program Interpreter, to end the current program, and then waits until receiving a signal that the current program has ended before continuing. If the Program Interpreter only signals that the current program has ended when the Robot Command Controller is waiting for such a signal, then the signal that the current program has ended can be placed in the list of Locked-State Events for the Robot Command Controller. Each instance of Locked-State Events must refer to a Control Object that exists within the

specification. No more than one instance of Locked-State Events can exist for any Control Object within a specification.

4.2 Concept Axioms and Inheritance

The foregoing definitions for the semantic concepts, composing the specification meta-model, were given in an informal manner in order to provide the reader with an intuitive understanding. More formal definitions are needed to facilitate automated support: 1) for verifying assertions that instances of a concept are indeed valid instances of the concept and 2) for evaluating whether or not a concept is a leaf concept in the specification meta-model. The meta-model addresses these requirements by providing axioms for each semantic concept within the concept hierarchy, and by allowing axioms from more general concepts to be inherited by more specialized concepts. This section explains the main principles of concept axioms and inheritance. Appendix A.1 gives a full specification of the axioms applicable for each semantic concept.

Each semantic concept within the specification meta-model can be circumscribed by a set of zero or more axioms. Each axiom consists of an axiom name and an axiom body. For example, consider the following axiom that applies to any instance of the concept Periodic Device Interface Object.

Axiom: **One, And Only One, Timer**

Let T be the set of all Timers whose sink is the given Periodic Device Interface Object. The cardinality of T must be one.

The axiom name, **One, And Only One, Timer**, is shown in **boldface** type and the axiom body is shown in *italics*.

Any instance of the concept Periodic Device Interface Object must also meet another axiom, as follows.

Axiom: **No Interrupt**

There must not exist an Interrupt whose sink is the given Periodic Device Interface Object.

These two axioms, **One, And Only One, Timer** and **No Interrupt**, must be satisfied by any instance of a concept that includes the concept Periodic Device Interface Object in its path along the concept hierarchy. Since Periodic Device Interface Object is not a leaf concept, these axioms would most likely be inherited by any of three child concepts: Periodic Device Input Object, Periodic Device Output Object, and Periodic Device I/O Object. The principles of axiom inheritance can best be illustrated through an example.

Consider the axioms that apply to the leaf concept Periodic Device Input Object, as shown in Figure 17. Figure 17 depicts a slice through the concept hierarchy, beginning with the concept Specification Element and ending with the concept Periodic Device Input Object. Each concept is illustrated with a circle. Concept inheritance is shown, as before, with a directed arc pointing from a child concept to its parent concept(s). For each concept, the names of the associated axioms are shown in a rectangle that is connected to the concept using a bi-directional arrow. The definition for each axiom can be found in Appendix A.1. Dashed, directed arcs connect the axiom rectangles to

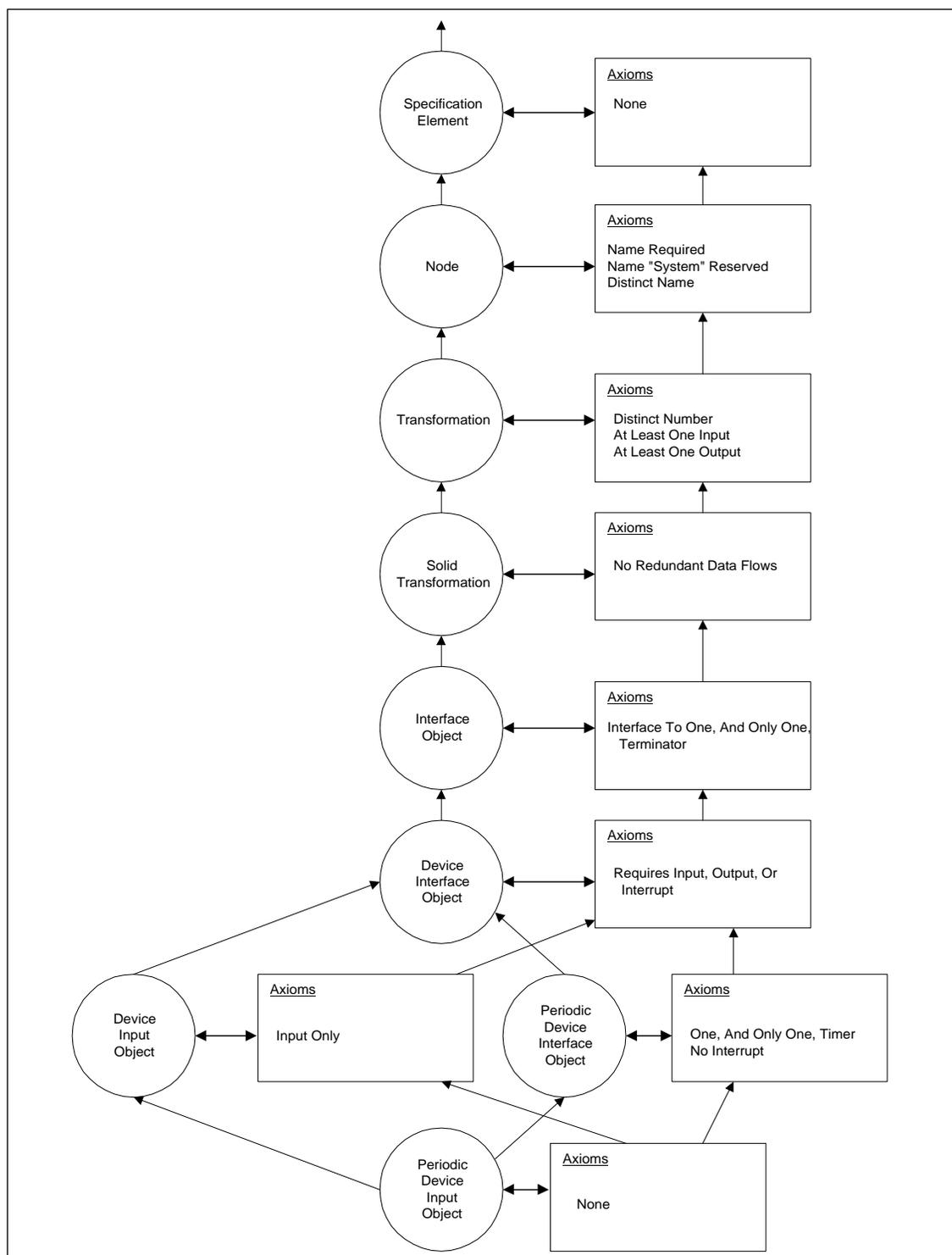


Figure 17. The Concept Hierarchy, Axioms, and Inheritance

illustrate that axioms are inherited in accordance with the concept hierarchy. The concept Periodic Device Input Object, even though it contains no axioms of its own, must satisfy the following axioms based on inheritance.

Node: **Name Required**
 Node: **Name "System" Reserved**
 Node: **Distinct Name**
 Transformation: **Distinct Number**
 Transformation: **At Least One Input**
 Transformation: **At Least One Output**
 Solid Transformation: **No Redundant Data Flows**
 Interface Object: **Interface To One, And Only One, Terminator**
 Device Interface Object: **Requires Input, Output, Or Interrupt**
 Device Input Object: **Input Only**
 Periodic Device Interface Object: **One, And Only One, Timer**
 Periodic Device Interface Object: **No Interrupt**

Similar groups of axioms can be composed for any concept in the specification meta-model. The group of axioms that apply to a given concept in the semantic meta-model comprise a formal definition for instances of the given concept. Any instance that satisfies the applicable axioms for that concept is a valid instance of the concept. Any instance that is asserted to be a specific concept and that then does not satisfy the applicable axioms for that concept is not a valid instance of the concept.

4.3 Classification Rules and Concept Classification

A given concept is a leaf concept in the specification meta-model if, and only if, there exists no other concept that inherits from the given concept. Each instance of a non-leaf concept represents a concept that is incompletely classified. All concepts in a given specification must be properly classified before a concurrent design can be generated. While an analyst or designer could make the necessary decisions, the

specification meta-model provides a basis for attempting to classify concepts automatically, using a set of classification rules deployed in an inference network to form a Concept Classifier.

A Concept Classifier should meet a number of objectives. First, classification decisions should be taken automatically, with human input elicited only where no automatic inference can be drawn. This requires that as much classification knowledge as possible be represented within the classification rules. Second, concept classification should assume that only the most minimal information might be provided on the input data/control flow diagrams. This means that a data/control flow diagram might be encoded using only the RTSA syntactic concepts (see Figure 5) within the specification meta-model: solid and dashed transforms, solid and dashed directed arcs, terminators, data stores, and solid, two-way arcs. The Concept Classifier, then, should be capable of beginning classification from this level of specification. Third, concept classification should make no assumption about the actual starting state of classifications within the input data/control flow diagram. This means that a particular input data/control flow diagram might contain concepts that are partially classified. The Concept Classifier, then, should be capable of determining which concepts are represented within a specification instance and should begin the classification from the correct point. These three objectives guide the formation of the concept classification rules described below.

The concept classification rules for the specification meta-model are defined using a form of **if-then** rule commonly employed in rule-based expert systems. The general form for a classification rule is as follows.

Rule: Classify A Concept

```

if
    a non-leaf concept is recognized and
    that concept satisfies the requirements of a more specific concept
then
    reclassify the concept as the more specific concept
fi

```

Here the rule name, Classify A Concept, is shown in underlined text, followed by the definition of the rule itself: **if** antecedent **then** consequent **fi**. A specific example of a classification rule to infer that an Interface Object is a Device Interface Object appears below.

Rule: Classify Device Interface Object

```

if
    the concept is an Interface Object and
    (the concept is the sink of an Input whose source is a Device or
    the concept is the source of an Output whose sink is a Device or
    the concept is the sink of an Interrupt whose source is a Device)
then
    classify the concept as a Device Interface Object
fi

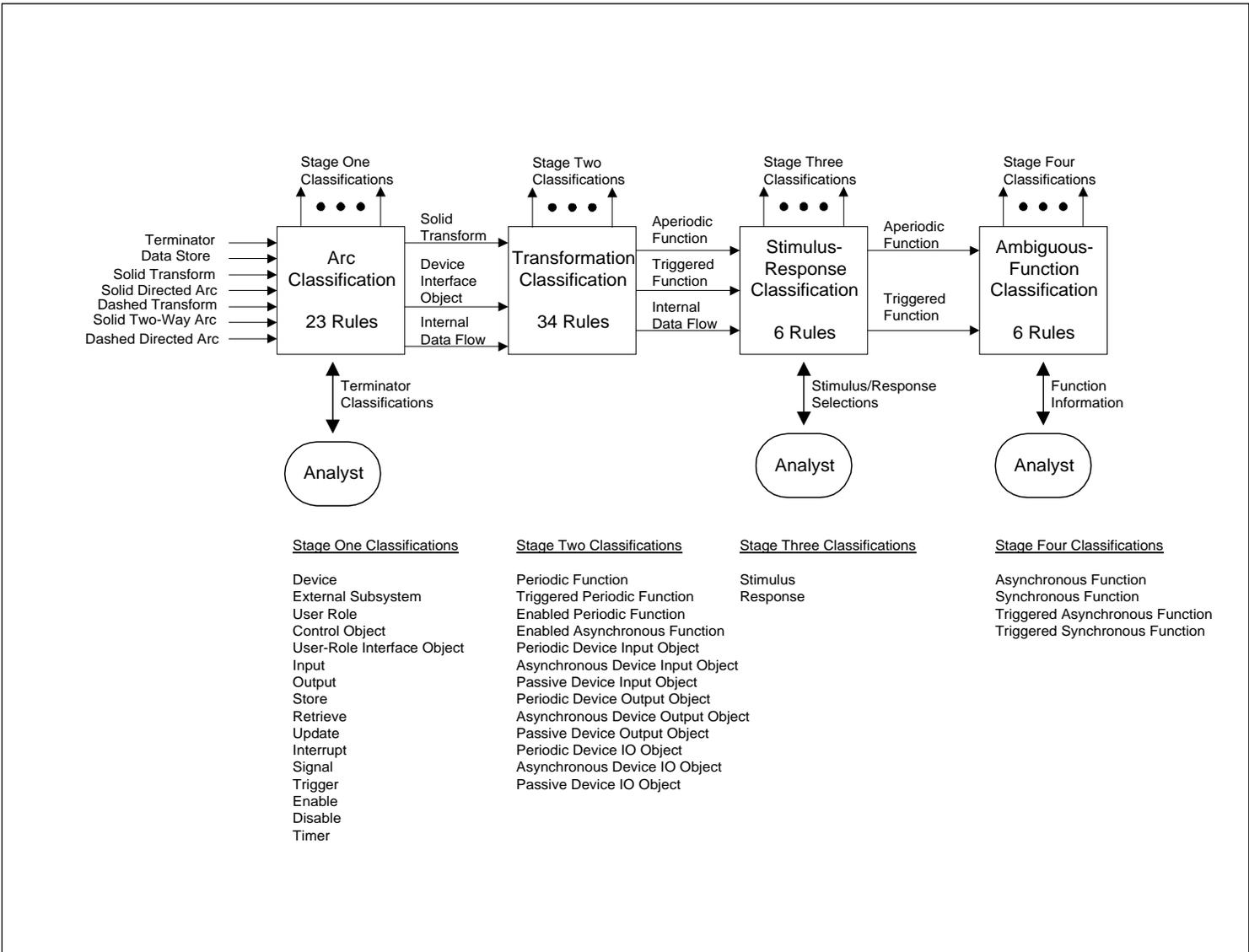
```

To meet the classification objectives outlined earlier, at least one classification rule is needed for each inheritance arc within the concept hierarchy. Appendix A.2 gives the complete set of classification rules for the specification meta-model.

The concept classification rules are organized in a four-stage inference network, as shown in Figure 18. Stage one of the network, the Arc Classification stage, attempts to fully classify all arcs on the input specification, also classifying Terminators and Transformations to the extent necessary to classify the arcs. The Arc Classification stage accepts concepts as minimal as those shown for its input: Terminator, Data Store, Solid Transform, Solid Directed Arc, Dashed Transform, Solid Two-Way Arc, and Dashed Directed Arc. These minimal concepts correspond to the syntactic elements for RTSA data/control flow diagrams. Stage one produces sixteen full classifications, as shown in the figure, two partial classifications (Device Interface Object and Internal Data Flow), and one non-classification (Solid Transform). The full classifications are available to later stages to help in making additional inferences. The partial classifications and non-classification can be viewed as direct inputs into stage two. During the Arc Classification Stage an analyst is consulted only when Terminator classification decisions are necessary. No means exist to infer the classification of terminators.

Stage two of the network, Transformation Classification, attempts to fully classify all transformations on the input specification. The Transformation Classification stage accepts the partial classifications and the non-classification from stage one and produces thirteen additional full classifications, as listed in the figure, and two partial classifications (Aperiodic Function and Triggered Function). The Internal Data Flow concept passes through the Transformation Classification stage and on to the third stage.

Figure 18. A Four-Stage Inference Network for Classifying Concepts



Stage three, Stimulus-Response Classification, resolves the classification of each Internal Data Flow on an input specification as either a Stimulus or a Response. The Stimulus-Response Classification stage produces only two additional full classifications (Stimulus and Response). In cases where the classification rules cannot classify an Internal Data Flow as a Stimulus or a Response an analyst is asked to make the classification. The Stimulus-Response Classification stage also passes the Aperiodic Function and Triggered Function concepts through to the final stage.

Stage four, Ambiguous-Function Classification, yields the remaining full classifications: Asynchronous Function, Synchronous Function, Triggered Asynchronous Function, and Triggered Synchronous Function. In cases where the classification rules cannot further classify an Aperiodic Function or Triggered Function without additional information, an analyst is asked to supply the necessary facts. For a Triggered Function the analyst might be asked whether or not the function completes during the triggering state transition. For an Aperiodic Function the analyst might be asked whether another function stops execution until the function completes or, if not, whether the function executes quickly when called upon.

The Concept Classifier requires a four-stage inference network because sometimes classification decisions regarding one concept depend on classification decisions regarding other concepts having been made fully. For example, classifying the range of device-interface objects possible within the specification meta-model can depend on the absence of certain concepts, such as Inputs, Outputs, Timers, and Interrupts. The

absence of these concepts cannot be assured until most classification decisions relating to directed arcs are taken. In addition, classifying data flows as Stimulus or Response depends, in some cases, on the existence of specific types of device-interface objects. In a like manner, deciding between asynchronous and synchronous functions depends on the full classification of all data flows.

The classification rules assume that any input data/control flow diagram must consist of at least the seven concepts, shown in Figure 5, as input to the Arc Classification stage; however, the input diagram can also consist of any concepts that are children to the seven minimal concepts, including any leaf concepts. These facts mean that: 1) classification rules must exist that begin with each of the seven minimal input concepts and 2) at least one classification rule must exist for each **is-a** path along the concept hierarchy. For the most part, each **is-a** path requires a single classification rule. Two exceptions are notable. First, since a user classifies each terminator as a Device, External Subsystem, or User Role, one classification rule can subsume the three corresponding **is-a** paths in the semantic meta-model. A second exception is the **is-a** path from Internal Data Flow to Stimulus. This path requires multiple classification rules because some special configurations on the data/control flow diagram are used to identify a Stimulus.

4.4 Information Elicitation

Certain information required to make design decisions might not be represented directly in an input data/control flow diagram. For example, specification addenda,

including Exclusion Groups, Aggregation Groups, and Locked-State Events, are not specified on a data/control flow diagram. In addition, use of the classification rules to automatically classify components from a data/control flow diagram might lead to the need for additional information in order to make design decisions. For example, if a Dashed Directed Arc becomes classified as a Timer, then a positive period must be supplied for the Timer. As another example, if a Solid Directed Arc becomes classified as an Input, then a maximum rate might be required for the Input. In general, when this additional information is needed but not found in the input specification, an analyst or designer must be consulted. The elicitation of the necessary information can be automated to the extent that: 1) the needed information can be identified automatically, 2) an analyst can be prompted for the necessary information, and 3) certain consistency checks can be performed on the information supplied by the analyst.