IEEE Standard for CASE Tool Interconnections—Reference Model for Specifying System Behavior

IEEE Computer Society

Sponsored by the
Software and Systems Engineering Standards Committee

1175.4™

IEEE
3 Park Avenue
New York, NY 10016-5997, USA

17 April 2009

IEEE Std 1175.4™-2008

IEEE Standard for CASE Tool Interconnections—Reference Model for Specifying System Behavior

Sponsor

Software and Systems Engineering Standards Committee
of the
IEEE Computer Society

Approved 10 December 2008

IEEE-SA Standards Board

Abstract: A reference model that provides a common interpretation basis by which tools can express and communicate the observable features of system/software behavior to users and to other tools is presented. This standard specifies a Conceptual Metamodel for understanding and describing the causal behavior for a system. The purpose of this Conceptual Metamodel is to express causal behavior and compositions of causal behavior in a model that integrates all observable operational features of a system into one behavior specification. This Conceptual Metamodel is useful for analyzing systems, for constructing particular system behavior models, and for using those models in the specification, design, and evaluation of engineered systems. It provides the necessary semantic elements for describing general hardware/software systems, including hardware-only, software-only, or mixed system components, and it allows these different types of components to be treated in a consistent manner, providing a basis for representing a wide variety of systems.

Keywords: behavior model, behavior specification, Computer-Aided Software Engineering (CASE) tools, engineering model, model-based testing, modeling principles, requirements, software specification, specification-based testing, system, system testing

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

PDF: ISBN 978-0-7381-5861-7 STD95857
Print: ISBN 978-0-7381-5862-4 STDPD95857

Introduction

This introduction is not part of IEEE Std 1175.4-2008, IEEE Standard for CASE Tool Interconnections—Reference Model for Specifying System Behavior.

The 1175™ family of standards

NOTE—References to "P1175.X" in this standard refer to members of the 1175 family of standards that were not yet approved at the time that this standard was published. [a]

This standard is a member of the 1175 family of IEEE standards. The members of this family include the following:

Standard number	Title
IEEE Std 1175.1™-2002 [B4] * †	IEEE Guide for CASE Tool Interconnections—Classification and Description
IEEE Std 1175.2™-2006 [B5]	IEEE Recommended Practice for CASE Tool Interconnection—Characterization of Interconnections
IEEE Std 1175.3™-2004 [B6]	IEEE Standard for CASE Tool Interconnections—Reference Model for Specifying Software Behavior
IEEE Std 1175.4™-2008	IEEE Standard for CASE Tool Interconnections—Reference Model for Specifying System Behavior
IEEE P1175.5™ ‡ **	Draft Standard for Computer-Aided Software Engineering (CASE) Tool Interconnections—Reference Data Metamodel for System Behavior Specifications

* The numbers in brackets correspond to those of the bibliography in Annex A.
† IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08854-1331, USA (http://standards.ieee.org/).
‡ This IEEE standards project was not approved by the IEEE-SA Standards Board at the time this publication went to press. For information about obtaining a draft, contact the IEEE.
** The title and description of IEEE P1175.5 are current as of the IEEE Std 1175.4-2008 publication date. Because the P1175.5 draft was not yet approved as of March 2009, this information is subject to change. For the most current P1175.5 information, please consult IEEE Xplore at http://ieeexplore.ieee.org/xpl/standards.jsp.

This family of standards replaces IEEE Std 1175-1991 [B3].[b] IEEE Std 1175-1991 was advanced to a full-use standard in 1994. It covered a number of closely related subjects, and the scope of material contained was able to serve a number of divergent interests.

This family of standards restructures and substantially augments the material in IEEE Std 1175-1991. It has been divided into several individually useful documents in order to facilitate its use by different communities of interest. These guides, recommended practices, and standards generally address issues involved in characterizing the kinds of interconnections that exist between a computing system tool and its environment. Although particularly intended to address the implementation and use of Computer-Aided Software Engineering (CASE) tools, the discussion of interconnections in this family actually has wider applicability to computing system tools in general, beyond only CASE tools.

[a] Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.
[b] Although approved in 1991, IEEE Std 1175-1991 was actually published in 1992 and is sometimes found referenced as IEEE Std 1175-1992. It appears in the standards numerical listing on the IEEE Xplore Web site (http://ieeexplore.ieee.org/) as IEEE Std 1175-1992, with the title IEEE Trial-Use Standard Reference Model for Computing System Tool Interconnections. In 1994, the term trial-use was removed from the title when the standard was approved for full-use status. The 1994 version, which was identical to the 1992 publication except for the title and minor editorial corrections, is not available on the IEEE Web site.

Four kinds of interconnections with a computing system tool are addressed: interconnections with organizations, users, platforms, and other computing system tools. Consideration of interconnections is important to understanding, selecting, implementing, and using computing system tools. Also, although many computing system tools do not need to communicate behavior descriptions of subject systems, their creators need to develop such behavior descriptions for the tools themselves.

A brief summary overview of each of the members of this family of standards is given in the following paragraphs. A more complete overview is available in IEEE Std 1175.1-2002 [B4], which provides an integrated overview of the members of the 1175 family of standards, and it describes the fundamental concepts that provide a basis for organizing the material.

IEEE Std 1175.1-2002, IEEE Guide for CASE Tool Interconnections—Classification and Description

IEEE Std 1175.1-2002 [B4] is a guide to the IEEE 1175 family of standards. It describes how these standards are intended to be used to accomplish the effective integration of computing system tools into a productive engineering environment and sets forth the fundamental concepts on which these standards are based. These concepts establish the integrating framework for the other members of this family of standards. IEEE Std 1175.1-2002 describes the scope of application of each member standard, the various issues addressed in each standard, and the interrelationships among the members of the 1175 family of standards.

IEEE Std 1175.2-2006, IEEE Recommended Practice for CASE Tool Interconnection—Characterization of Interconnections

The IEEE Std 1175.2-2006 [B5] recommended practice presents four contexts for a computing system tool's interconnections that offer insight into the operational problems of interconnecting computing system tools with their environment. This recommended practice establishes recommended collections of standard contextual attributes describing relationships between a computing system tool and its organizational deployment, its human user, its executable platform, and its peer tools, as illustrated in Figure a. These contextual attributes are of the "news-story" form that includes: who, what, when, where, and why. The values of these contextual attributes are references to organizational, industrial, and professional standards. By assisting users to reach a clear understanding of the context of operation for a computing system tool, this recommended practice contributes to the effective implementation and application of computing system tools.

IEEE Std 1175.3-2004, IEEE Standard for CASE Tool Interconnections—Reference Model for Specifying Software Behavior

IEEE Std 1175.3-2004 [B6] is an expansion of Part 3 of IEEE Std 1175-1991. It focuses specifically on a common set of modeling concepts found in commercial CASE tools for describing the operational behavior of a software product, and it provides a formal, logical model for describing this behavior. IEEE Std 1175.3-2004 also defines a Semantic Transfer Language (STL) for communicating software behavior descriptions from one tool to another. A notable feature of the STL is its design for human readability, which makes STL text files suitable for use in software design reviews by users unfamiliar with computing system tool diagramming notations. In addition, the design of the STL syntax readily permits analysts to prepare and edit STL descriptions using a text editor or word processor.

To permit backward compatibility with Part 3 of IEEE Std 1175-1991, IEEE Std 1175.3-2004 makes no changes to the STL syntax or to the rules for conformance to this syntax as originally defined in that standard. However, some aspects of the 1991 syntax that were previously left as user-defined have now been specified in order to increase the consistency and reliability with which the STL may be used for exchanging software specification information. In addition, improvements have been made in how the STL syntax is defined and explained. Finally, the STL Interconnection Profile has been replaced with more straightforward, "user-friendly" tabular and comma-separated-value formats to define a Tool Interconnection Profile that can serve the same purpose as the original form of the profile.

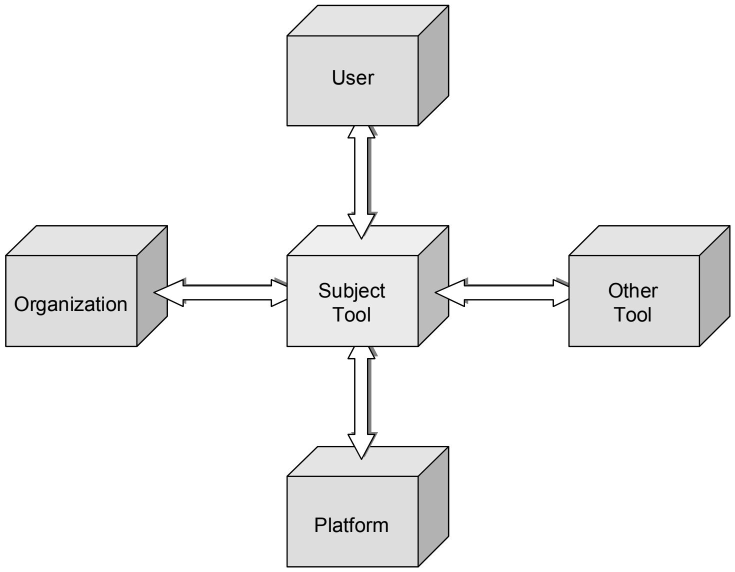


Figure a

IEEE Std 1175.4-2008, IEEE Standard for CASE Tool Interconnections—Reference Model for Specifying System Behavior

IEEE Std 1175.4-2008 encompasses the description of the types of the computing systems supported by IEEE Std 1175.3-2004 [B6], but it goes further, providing a basis for representing a wider variety of systems. Specifically, IEEE Std 1175.4-2008 provides the necessary semantic elements for describing general hardware/software systems, including hardware-only, software-only, or mixed system components, and it allows these different types of components to be treated in a consistent manner.

IEEE P1175.5, Draft Standard for Computer-Aided Software Engineering (CASE) Tool Interconnections—Reference Data Metamodel for System Behavior Specifications

This standard defines a Data Metamodel for system behavior specifications. Figure b illustrates one use for such a behavior specification metamodel in the context of supporting information transfer from one user's tool to another user's tool. The Data Metamodel provides explicit definitions of typed data elements, information representations, and relationships with which behavior models for subject systems can be instantiated. These elements, representations, and relations serve to reify the Conceptual Metamodel for system behavior specification described in IEEE Std 1175.4-2008.

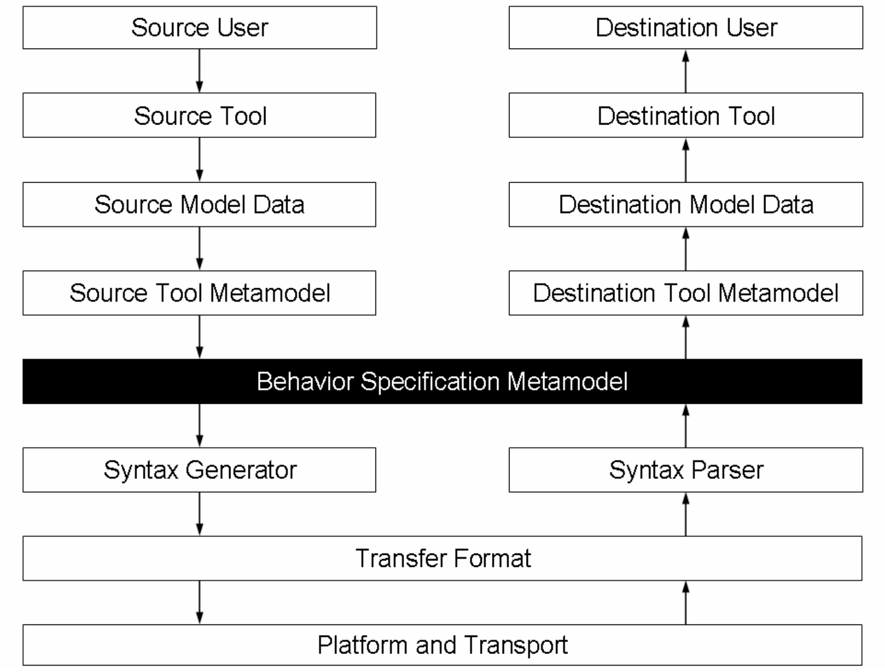


Figure b

When multiple tools are being used to describe a system, each may maintain its own information metamodel. However, as depicted in Figure b, to share information about a subject system, each tool must map its own individual metamodel into a common Behavior Specification Metamodel.

Notice to users

Laws and regulations

Users of these documents should consult all applicable laws and regulations. Compliance with the provisions of this standard does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

This document is copyrighted by the IEEE. It is made available for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

Updating of IEEE documents

Users of IEEE standards should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Standards Association Web site at http://ieeexplore.ieee.org/xpl/standards.jsp, or contact the IEEE at the address listed previously.

For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA Web site at http://standards.ieee.org.

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/reading/ieee/updates/errata/index.html. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this standard was submitted to the IEEE-SA Standards Board for approval, the 1175.4 Working Group had the following membership:

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Contents

IEEE Standard for CASE Tool Interconnections—Reference Model for Specifying System Behavior

1. Overview

1.1 Scope

Most inter-tool data transfer standards deal with protocol and syntax of the transfer, with a shared semantic basis assumed. This standard provides an explicitly defined metamodel (and meta-metamodel) for specifying system and software behavior. It defines a semantic basis of observables that allows each tool, whatever its own internal ontology, to communicate facts about the behavior of a subject system as precisely as the tool's metamodel allows. Conventional tool model elements are reduced into simpler, directly observable fact statements about system behavior. This metamodel is much expanded over the original metamodel for software behavior in Part 3 of IEEE Std 1175™-1991 [B3].[1]

1.2 Purpose

This reference model provides a common interpretation basis by which tools may express and communicate the observable features of system/software behavior to users and to other tools. Tools incorporating this metamodel in their import/export facilities enable engineers to interconnect best-in-class analysis and specification tools for integrated problem solving. Another feature of this metamodel is that it

[1] The numbers in brackets correspond to those of the bibliography in Annex A.

provides a specification that is directly testable. Finally, the provision of an explicit meta-metamodel enables tool builders to extend the reference metamodel for particular purposes.

1.3 Applicability

A fundamental notion in this standard is that the engineering specifications (technical descriptions) of a system are models of that system. Each model characterizes some aspect of the system from a particular viewpoint to serve particular engineering needs. Functional and physical elements associated with a particular aspect of the system are described and organized into models of all necessary aspects of the subject system.

The aspect of systems for which the model described in this standard may be used is the causal (stimulus-response) behavior and affective (functional or historical dependence) behavior of systems. (These behaviors are discussed in more detail in 6.3.1.) The class of systems for which the model described in this standard may be used are engineered systems that interact dynamically and deterministically with some part of the external world in which they exist, for effecting changes in that external world. Such systems have material existence, they have measurable static and dynamic properties, and they consume energy to provide their effects.

The class of engineered system addressed by this standard includes those that take action based on their sense of properties or events in their environment. Such systems monitor their environment and predictably perform expected actions based on the environment they perceive. Their only causal behaviors might be an OFF-ON state change and an ON-OFF state change controlled by a power switch. Examples might include a patient-monitoring system, an intrusion detection system, or an automatic home coffee-maker. These are primarily passive systems that do not rely on external agents to stimulate their performance.

This class of engineered system also includes those that do not take action unless directed to do so by a stimulus from their environment. Examples would include an information system storing input data, responding to edits, and producing reports on request, or a remotely commanded sensor system that senses phenomena in the local environment, stores selected digital data streams, transmits data records, or purges old data records. These are reactive systems that do rely on external agents to stimulate their performance.

Numerous models and modeling frameworks are currently used to understand, develop, and document the detailed structural and transformational relationships in the design of such systems. The interaction behavior, which is the only perceptual basis for fitting the subject system into its environment and verifying that the subject system instance conforms to that basis, is sometimes lost or difficult to reconstruct from those models.

This standard specifies a Conceptual Metamodel for understanding and describing the causal and affective behavior for a system. While many tabular, graphical, and animated representations of this information can be used for human presentation and interaction within documents, drawings, and tool media, characterizing those representations is not within the scope of this standard.

This standard does not address other models of systems that facilitate the management, development, production, and use of engineered systems. Such models include reliability and availability models, safety models, security models, physical models of form and fit, cost and schedule estimation models, technology readiness models, failure and fault models, manufacturability and production models, and environmental assault and survivability models. It is a system engineering concern to understand the partitioning of system aspects among such models, the relationships between these separate models, and the relationships between these models and the subject system.

1.4 Engineering models and metamodels

Engineering models can be characterized by what elements of the system are selected for description in the model, by the forms and patterns used for model description, and by the kinds of inferences that can be made from the model.[2] Within the scope of those selected elements, forms, and patterns, the model becomes a representation (to some degree of fidelity) of the actual system. With the addition of appropriate methods of evaluation and prediction, the model can become a useful engineering tool.

The dependability of such a tool requires standardization of modeling forms and patterns, an unambiguous relationship to identifiable system elements, repeatable inferences from the model, and a coherent interpretation of predictions. The set of syntactic and semantic rules that define these things is called a *metamodel*.

Models have a purpose that is served by the choice of descriptive elements in the model, the questions that can be posed to and answered with the model, and the methods by which inferences are made. A model can be expressed in a variety of representations. The choice of representation depends on that representation's capability to represent the things and relationships being modeled, the adequacy of inference mechanisms for extracting information from the model, and the ease of creating the model and extracting inferences.

The purpose for a system behavior model is to represent the net, externally observable behavior of a system. It is thus centered on the interactions of a subject system with its environment. The intended use of this model is to understand the services it provides to its external environment and the obligations of the external environment that enable successful operation of the system. This is a pure black-box specification of a system that is fully distinguished from its environment.

The reference model in this standard is a particular Conceptual Metamodel for the description of system behavior, viewing system behavior as the delivery of application value into an application domain through the aggregate of all system responses to internal and external stimuli.[3] It is focused on the development and expression of the total mapping of sequences of stimulus interactions to sequences of response interactions brought about by the operation of a subject system.

This reference model contains a small number of fundamental concepts abstracted directly from the observable instances and patterns of system behavior. When cast into a corresponding Data Metamodel, these concepts and their relationships provide the basis for interpreting a data model of system behavior and for inferring the possible sequences of observable behaviors of the subject system.

A system behavior model focuses on the interactions that occur between elements in the application and the subject system. It models the ways in which the system senses properties and events in the application domain and the ways in which the system causes desired changes in that domain. Generally, a system behavior model does not contain explicit identification of every mapping of possible stimulus sequence to resulting response sequence. Rather, it defines the totality of patterns that encompass all possible occurrences.

System behaviors are a part of *system requirements,* as they state requirements on the system's interaction with the application domain. System requirements could also be called *product requirements, system specifications,* or *product specifications.*

Things to be modeled in the system's interaction with the application domain include the following:

— Selection of relevant external interactions that are shared with the application domain

— Selection of temporal granularity (modeling in "point time" or in "continuous time")

[2] The system modeling perspective for a tool is described in 5.2 of IEEE Std 1175.2™-2006 [B5].
[3] This interpretation of behavior is adapted from definition 2.b for *behavior* found in Dictionary.com Unabridged (v 1.1).

— Representation of selected interactions in terms of physical, logical, and temporal characteristics

— Causal relationships between occurrences of stimulus interactions and response interactions

— Affective relationships between input and output interaction contents

— Affective relationships sequences of behaviors

Prior to the existence of a system, these are behavior specifications to be met. Following development of a system that purports to comply with the model, these provide one basis for testing and accepting the system (through verification).

A common failure in system behavior modeling is the assumption that the only stimulus and input interactions that occur are those that are desired. Neither the model nor the system will be robust (predictable) if this assumption is invalid.

Annex B provides a brief description of several other engineering models commonly used in system development and describes their relationship with the system behavior model. The following models are addressed:

— Stakeholder requirements model

— System use model

— System mechanism model

— System implementation model

— System verification model

— System validation model

1.5 Audience

The audience for this standard consists of systems analysts, system designers, system modelers, product specifiers, product architects, product designers, product design verifiers, specification validators, software testers, simulation developers, and specialists in the verification and validation of models and simulations.

1.6 Conformance

The reference model for specifying system behavior is described in a Conceptual Metamodel, as explained in Clause 5. The Conceptual Metamodel requires a Data Metamodel as the tool by which a system modeler can express, document, store, and analyze a behavior model for a particular system. A Data Metamodel enables modelers to construct a model of some system’s behavior in accord with the Conceptual Metamodel.

The various observables, relationships, and fact types identified in the Conceptual Metamodel (see 5.1) constitute the core requirements to be satisfied by a Data Metamodel. That is, any proposed Data Metamodel shall provide an explicit representation of the Conceptual Metamodel elements so that model facts can be recorded and manipulated.

A Data Metamodel conforming to this Conceptual Metamodel shall provide representations and structures of data elements, data values, and expressions for recording and communicating the conceptual elements identified in Clause 7. It shall also provide model verification rules to be applied over populated data structures for evaluating completeness and consistency of a particular system behavior model with respect to the Conceptual Metamodel.

1.7 Organization of this standard

This document contains seven clauses and two annexes. The following annotations provide a roadmap to the contents:

Clause 1 describes the purpose and scope of this standard and also provides conformance criteria. It distinguishes system behavior models from other types of engineering models.

Clause 2 lists the references necessary for implementing this standard.

Clause 3 provides definitions of terms.

Clause 4 identifies the requirements to be met by a reference model for system behavior, and some of the capabilities it would provide.

Clause 5 distinguishes the Conceptual Metamodel defined in this standard from the Data Metamodel defined elsewhere, and it describes the complementary roles of each in providing the reference model for specifying system behavior.

Clause 6 identifies and elaborates three general principles on which the Conceptual Metamodel worldview in this standard is based and explains the nature and role of each element in the Conceptual Metamodel. It provides an inference concept for predicting observable unit behavior. It also provides a composition structure for relating behaviors of an assembly of units to behaviors of the units composing that assembly.

Clause 7 summarizes the elements and relationships within the Conceptual Metamodel for system behavior. This clause also describes how this standard is to be applied.

Annex A provides informative references.

Annex B provides a brief description of several other engineering models.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

The following glossary standard shall be used, when applicable, for tool or software terms not defined in this standard.

IEEE Std 610.12™, IEEE Standard Glossary of Software Engineering Terminology.[4, 5]

[4] IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08854-1331, USA (http://standards.ieee.org/).
[5] The standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

3. Special terms

For the purposes of this document, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms* [B2] or IEEE Std 610.12[6] should be referenced for terms not defined in this subclause.

Other relevant terms defined in Clause 2 of IEEE Std 610.12 include: **CASE**, **interface**, **semantics**, **software tool**, and **syntax**.

The following additional definitions describe specific terms as used within the context of this standard.

3.1 action: A statement of causal and affective relationships in a behavior model linking particular stimulus interactions to particular response interactions and changes within a unit under a certain set of conditions on a unit's lifeline.

3.2 affective relationship: A functional dependency between prior input interaction occurrences and later output interaction occurrences in a behavior pattern.

3.3 assembly: A collection of units in which ports on different units are compatibly interconnected so behaviors of one unit can influence behaviors of another unit by means of interactions occurring through interconnected ports.

NOTE—An *assembly* may be regarded as behaviorally equivalent with a unit. The ports of the unit are just those ports of the assembly units that are not internally interconnected. The behavior of the unit is a composition of the shared behaviors of the assembly units.[7]

3.4 available interaction: An interaction that is allowed by one unit and controlled by another unit.

3.5 behavior: The aggregate of all unit responses to internal and external stimuli. *See also:* **behavior pattern.**

3.6 behavior pattern: A relationship that maps a sequence of stimulus interactions to a sequence of response interactions. *See also:* **affective relationship, causal relationship.**

3.7 behavior state: A state which represents the partitioning of all unit behavior patterns into (possibly overlapping) subsets of behavior patterns that can be elicited from a unit at a particular point in time.

3.8 boundary: A collection of all interface ports between a unit and its environment, characterized by the input and output interactions that it allows.

3.9 causal relationship: An existence dependency between the stimulus of a behavior and the various responses that are caused to occur.

3.10 condition: A logical predicate involving one or more behavior model elements.

NOTE—A *condition* predicate may be a statement about some characteristic of a model element, a statement about a relationship between model elements, or a logical combination of such statements.

[6] Information on references can be found in Clause 2.
[7] Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement the standard.

3.11 coordinated interactions: A pair of interactions in which the first interaction is generated by a unit as a stimulus to some other unit, in expectation that a second interaction will be generated as a response from that other unit (or even a third unit) to benefit the initiating unit for a particular purpose. *See also:* **obligation.**

NOTE—One unit controls the initiation of this exchange, subordinating the other unit(s) to itself. Such a subordinating exchange is possible even if the subordinated unit generated the stimulus that triggered the initiating unit's behavior.

3.12 event: A singular moment in time at which some perceptible phenomenological change (energy, matter, or information) occurs at the port of a unit.

3.13 guarantee condition: A statement of the constraints that will be satisfied by output interaction occurrences and the next property state as a result of the occurrence of a particular behavior pattern.

3.14 guard condition: A statement of the circumstances (input interaction occurrences and property state) that allow a stimulus to cause the occurrence of a particular behavior pattern.

3.15 interaction: An identity of phenomena existing over some period of time at the interface between two units, caused by one unit and affecting the other unit. *See also:* **available interaction, coordinated interactions, response, stimulus.**

NOTE—The identity is expressed in relevant phenomenological terms. Generally, an interaction identity can be categorized as energy transfer, matter transfer, or information transfer.

3.16 interaction alias: A relationship in a build specification that matches an interaction instance in one port with an interaction instance in another port.

NOTE—The *interaction alias* is used to indicate identity or compatibility of matching interactions at two ports of a port couple or a port alias. In a port alias, both interactions have the same direction. In a port coupling, the interactions have opposite directions. This provides an element of "model bookkeeping" necessary for permitting models developed independently for different units to be integrated into a unified composite model.

3.17 lifeline: Any historical sequence of behaviors through which a unit is manipulated by external stimuli.

3.18 obligation: A collaboration pattern of interaction between two units used when one unit is not independently capable of providing its service behavior without the assistance of the other unit.

3.19 port: A n interface on a unit's boundary that allows a subset of the interactions that can be exchanged with another unit.

NOTE—Interactions at a *port* have a common phenomenological basis by which they are exchanged. Occurrences of interactions in one direction are serialized at a port.

3.20 port alias: a replacement relationship in a build specification that identifies a port of one unit with a port of a subunit and indicates that interactions at the two ports can be paired identically or compatibly.

NOTE—A *port alias* is used to indicate identity or compatibility of two ports at different levels of assembly in which the port in a lower level unit of assembly serves as a realization of a port a higher level unit of assembly level. This provides an element of "model bookkeeping" necessary for permitting models developed independently for different units to be integrated into a unified composite model. Compatibility criteria are satisfied by valid pairings.

3.21 port couple: An interconnection relationship in a build specification that identifies a port of one unit with a port of another (structural peer) unit and indicates that interactions output from one are input to the other, and vice versa.

NOTE—A *port couple* defines a one-to-one matching of the two ports defining opposite sides of a particular interface. Ports match identically if all output interaction occurrences at one port match are identical with all input interaction occurrences accepted at the other port. Ports match compatibly if interaction occurrences satisfy specific compatibility criteria.

3.22 property: A measurable phenomenological characteristic of an interaction.

3.23 property state: A state comprising properties associated with a unit that are defined by functional dependence on the properties of past interaction occurrences and that affect the properties of future output interaction occurrences.

NOTE—A particular *property state* of the unit is defined at a point in time by the values of the internal state properties.

3.24 response: A reaction of the unit to a specific stimulus, depending on the current conditions on the unit's lifeline.

NOTE—A *response* might be the occurrence of internally controlled interaction(s) affecting the environment (external response), or it might be the occurrence of an internally controlled change to the unit's state (internal response).

3.25 state: A characteristic of a unit (usually composite in structure and multi-dimensional) expressing the cumulative effect of all previous unit interaction occurrences, in terms of which the delayed affective relationships of the unit can be evaluated. *See also:* **behavior state, property state.**

NOTE—A *state* is a shorthand representation for the unit's interaction occurrence history. Ascribing "state" to a unit implies that it has a capability to modify future behaviors as a result of past interactions with its environment.

3.26 state invariant condition: A statement of constraints or relations that can be used to distinguish a subset of particular property states that all satisfy the condition.

3.27 stimulus: Whatever causes a unit to exhibit an occurrence of a behavior pattern in the unit's repertoire; something causing or regarded as causing a response.

NOTE—A *stimulus* might be the occurrence of an externally controlled interaction affecting the unit (external stimulus), or it might be the occurrence of an internally controlled event (internal stimulus).

3.28 subject unit: A unit that is the subject of a behavior modeling.

NOTE—A *subject unit* is segregated from a collection of units that constitute the environment in which its behaviors are solicited and exhibited.

3.29 unit: A distinguishable architectural unit with individual identity, boundary, and behavior that is observable through interactions with other such units.

NOTE—A *unit* may be a system, a subsystem, a system element, an assembly of components, or a primitive component in a system.

4. System behavior model requirements

As defined in this standard, the reference model for system behavior satisfies the following requirements:

— Application scope: the reference model is capable of specifying behaviors for a useful range of interesting systems.

— Declarative form: the reference model is restricted to the specification of causal and affective behavior effects, without the specification of mechanisms for producing that behavior.

— Predictive ability: the reference model supports the inference of required and allowed behaviors from the declarative patterns in the model.

— Test ability: the reference model expresses all behavior in terms of observable interactions.

— Life cycle role: the reference model serves specific life cycle purposes in the process areas of requirements definition, technical design, and verification.

— Common interpretation: the reference model provides a context of interpretation that allows multiple users (and tools) to make similar inferences for questions about the subject system behavior.

This clause addresses the needs and constraints associated with each of these requirements.

4.1 Application scope

The reference model for causal and affective behavior presented in this standard describes reactive systems that exchange information, energy, and matter with other systems in their operational environment. The systems of concern in this standard have physical realizations and exhibit repeatable, or at least predictable, behaviors that are used to bring about specific intended effects in their environment. Such systems provide value into an application domain through the exchange of interactions between the subject system and other hardware, software, or bioware systems.

While modern systems are realized with considerable complexity, their specification descriptions are often biased to model some system features in far more detail than others. Risk considerations drive the focus onto the primary value-producing areas of complexity in the system. Furthermore, the adoption of *de facto* design and implementation standards could minimize the level of concern for other feature areas. For real-world systems, a *most-like* determination suggests the use of particular specification models with particular focus areas.

— The value of information management systems tends to focus on the organization and storage of their data content and the business and editing rules for preventing the acceptance of invalid information. The external interactions of these systems are stylized through the definition of screen sets and report sets.

— The value of control systems tends to focus on the detection of complex structures of input signals, the interpretation of certain inputs within the sequence of previous inputs, and the timed or timely generation of complex structures of output signals. The data content of these systems is often of the form of parameter sets and logs.

— The value of simulation systems tends to focus on the accurate rendition of output data streams with respect to specified model parameters and initial conditions.

— The value of desktop office tools tends to focus on the construction and manipulation of virtual objects (e.g., lists, documents, diagrams), with storage, recall, and transfer to various output media.

— The value of networks tends to focus on rapid, uncorrupted point-to-point information transfer through complex node and link topologies, with extensive messaging choreography to manage end-to-end reliability and protection.

While focused and detailed specification models address specific features, they usually omit other features altogether. Specification frameworks have emerged to identify the more inclusive set of issues that should be included in specification. The purpose of the Conceptual Metamodel defined in this standard is to express causal and affective behavior and compositions of such behavior in a model that integrates all observable operational features of a system into one behavior specification. In practical use, this specification should be complemented with the additional detail available in other specialized models, but it provides the framework in which those additional details fit and have an observable meaning.

The stakeholder requirements identify what accomplishments are to be wrought in the application domain through the operation of the subject system. The system behavior model captures the pre-design analyses and decisions about how the subject system will interact with that application domain, and what rules need be satisfied by those interactions so that it will have its intended effect. Developing and recording the description of this causal and affective behavior provides several value-added opportunities for assurance.

— Constructing a causal and affective behavior model for a subject system encourages perception of the subject system in its physical environment. Often, important design bases are understood earlier in a life cycle when accounting for all interaction ports, multiplicities of architectural units, potential interactions and functional concurrency, and timing and duration expectations.

— Combining a detailed causal and affective behavior specification with assumptions of the environment, it is possible to investigate whether the specified subject system can be expected to actually satisfy the stakeholder requirements.

— Comparing an assembly of component causal and affective behaviors against the net causal and affective behavior of the assembly provides evidence for soundness of decomposition; it can also elucidate assumptions that need confirmation and ambiguities that need resolution.

— Organizing the required observables of system operation into a causal and affective behavior model provides a direct basis for inferring interaction streams to stimulate, sensitize, probe, and observe all the expected behaviors of the subject system. This allows the development of test cases and test suites in which test cases are traceable by construction to the specifications. The behavior specification can provide a direct, countable reference base for arguing specification-based adequacy of the test suite.

The normative content in this standard is neutral with respect to the scale of the object whose behavior and structure is modeled. Rather than using terms such as *system*, *subsystem*, or *component*, the object of attention is abstracted to *architectural unit*, or simply *unit*. A unit is always embedded in some operational context or environment and exhibits behavior through its interactions with that environment. A unit also may have a composition in terms of interconnected units whose individual behaviors aggregate to some extent as the behavior of the assembly.

The reference model is focused on the specification of a unit in isolation, with the identification of all the behavior patterns by which it delivers its capabilities into a context of other units.

4.2 Declarative form

A behavior specification model identifies the exchanges of matter, energy, and information between the subject system and its external environment, and it specifies the rules and relationships among these exchanges so that system behavior satisfies the stakeholder requirements. (Practically, system behavior can intentionally satisfy only the stakeholder requirements as understood and documented by the stakeholders.) It is inappropriate for the behavior specification model to describe a mechanism for these behaviors, but it should identify what behaviors the subject system delivers into the application domain through specified interactions, relationships, and properties. Thus, the behavior modeling approach should be declarative. Many different mechanisms may be designed to produce the specified behaviors, differing in their satisfaction of any number of other design criteria. The chosen mechanisms would themselves be described and characterized by a variety of mechanism models. Design constraints are not part of a behavior model.

From the perspective of the application domain, the system behavior model is imperative. Its specified interactions and rules of behavior provide a means by which the requirements in that domain can be made true. With respect to the subject system itself, however, the behavior model defines and describes only what is visible in the application domain. It does not describe any mechanism for generating that behavior, only how to recognize the behaviors.

The behavior model is to be restrictively interpreted. It is a model for which some scope of behaviors is selected for inclusion. For those selected behaviors, it is a description of what is controllable and observable in the subject system. A permissive interpretation, on the other hand, would assume that whatever is not prohibited is allowed; such a specification would describe only constraints. A permissive interpretation has an unbounded scope of verification and cannot be the basis for engineering assurance in the development of the subject system.

The behavior model is declarative in that its meaning is inferred from the elements in that model, their definitions, and their relationships. There is no meaning implied by any sequencing of the model content.

4.3 Predictive ability

A system interacts dynamically with some part of the external world in which it exists to effect changes in that external world. Let $E(t_0)$ and $S(t_0)$ describe the environment and the system, respectively, at time t_0.

The environment E includes objects that exist even in the absence of the system S, as well as objects that might exist because of the system. From a stakeholder requirements viewpoint, there is some collection of environment objects that includes those objects that interact directly with system S, as well as other objects that either indirectly affect S or are indirectly affected by S. From a behavior specification viewpoint, all that is perceivable in the environment E is the set of interactions that pass between E and S.

As interactions between E and S occur over a period of time, both the environment and the system change, from $E(t_0)$ to $E(t_1)$ and from $S(t_0)$ to $S(t_1)$, respectively. Figure 1 illustrates the synchronous evolution of environment and system on parallel lifelines as they interact through a collection of shared phenomena P.

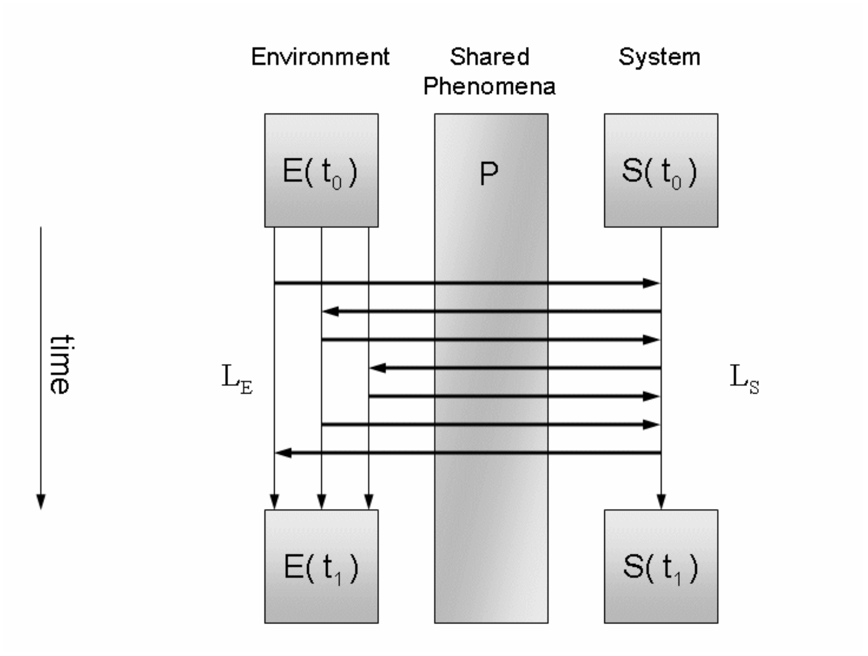


Figure 1—Representing temporal transformations in the world

The environment E evolves under the influence of the interactions from S in response to the interactions issued from E. Denoting the evolutionary lifeline path of E by L_E, the situation can be expressed as an evolutionary transformation of $E(t_0)$ to $E(t_1)$ (as denoted in Figure 1):

$$L_E : E(\, t_0\,) \rightarrow E(\, t_1\,)$$

(The transformation L_E is actually a composite of transforms across the lifelines of the objects in E.) A stakeholder requirements model aims precisely at identifying and describing those objects included in E

and at defining particular lifeline trajectory points such as $E(t_0)$ that exist there. The system use model then aims at characterizing some of the transformations L_E that could be used to connect those identifiable points.

The system S evolves under the influence of particular sequences of particular interactions from E. Denoting the evolutionary lifeline path of S by L_S, the situation can be expressed as an evolutionary transformation of $S(t_0)$ to $S(t_1)$ (also denoted in Figure 1):

$$L_S : S(t_0) \rightarrow S(t_1)$$

The behavior model for the system aims at identifying and describing those interactions that are possible through phenomena in P and at defining the required cause-effect relationships that can be used to steer the evolution of $E(t_0)$ to $E(t_1)$. Any particular L_S for the system results from a composition of particular behaviors in particular sequences. There is considerable freedom in the specification of both the L_S trajectories and the basic behaviors in them, even when there are constraints on the L_E trajectories. Such freedom should be exploited to optimize other behavior-related properties of the system, to include reliability, safety, and testability.

Figure 1 shows that a specific L_E and L_S are the same, since both are expressed as the same sequence of interactions. However, the set of all L_E is not the same as the set of all L_S. In an idealized world, the two sets would be the same. In a real world, they cannot be the same. The difference is due to the difference between what is desired and what is possible in the possible interactions between E and S. The set of possible L_S differs from the set of desired L_E due to possible "errors" in the interactions and interaction sequences emanating from E. (These possibilities include erroneous interaction content, erroneous interaction timing, erroneous ordering of interactions, and missing interactions.) A comprehensive analysis would consider both the desired set of L_E as well as a "corrupted" set of L_E to assemble the complete set of interactions and sequences that could be expected in the set of L_S.

NOTE 1—Interaction errors and corruption are real world issues. Expanded functionality is required in environment and system objects to discriminate, reject, respond, and recover from their presence. The behavior specification should not assume the absence of these effects, so the metamodel must provide for representing these issues.

Similarly, the actual complete set of L_E differs from the set of desired L_S due to possible errors in the interactions and interaction sequences emanating from S. Thus, objects in E should be prepared to evolve under the possible, even if unexpected, interactions and sequences from S.

In many interaction sequences, there is a kind of dance in which an environment object and the system alternate turns in generating interactions to one another. This has the effect of synchronizing the two unit lifelines as both units experience the interactions concurrently with the other at specific points in time. The time synchronization between the two is private, and it provides a simple untimed protocol for providing system effects into the environment. This is inadequate when interactions are required at specific times or on specific interval measures, or when the two units are required to synchronize with other environment object lifelines. More generally, some form of common time measurement is necessary.

It is assumed that a notion of time, continuous to some granularity, can be ascribed to the system as it evolves through its lifeline. Units of time can be marked or noted and the interval of time passage can be determined by various means. Some external interaction behaviors usually exist for synchronizing the unit's timekeeping capability with time points in the unit's environment. In addition, the system might have nonbehavioral requirements for stability and lifetime of timing capability.

From a modeling perspective, this assumption allows the notion of temporally synchronized traversal of lifelines by environment objects and by the system.

It is a complex task to analyze the set of all required L_E that fulfill the stakeholder requirements in E, and it is a different complex task to determine for S the complete collection of causal and affective behaviors from which the corresponding L_S can be composed.

System behavior is the aggregate of all system responses to internal and external stimuli. It is a mapping of sequences of stimulus interactions to sequences of response interactions. The goal of the reference behavior model is to enable the description of all causal and affective behaviors.

NOTE 2—The practicality of defining a software program as an explicit mapping of input sequences to output sequences with no local variables (data state) was demonstrated in the Lucid programming language (Wadge and Ashcroft [B8]).

To describe a particular system behavior requires the following:

— The description of a particular $S(t_0)$

— The representation of a particular L_S

— The prediction of $S(t_1)$ from $S(t_0)$ and L_S

Correspondingly, the reference model must provide the following:

— A means to express a model $M(t_0)$ corresponding with $S(t_0)$

— A means for composing declarative cause-effect patterns into an evolutionary path L_M in M that corresponds with an evolutionary path L_S in S

— An inference mechanism that exhibits the following relationship:

$$L_M : M(t_0) \to M(t_1)$$

Figure 2 illustrates this correspondence relationship between the predictive capability of the reference model M and a system S.

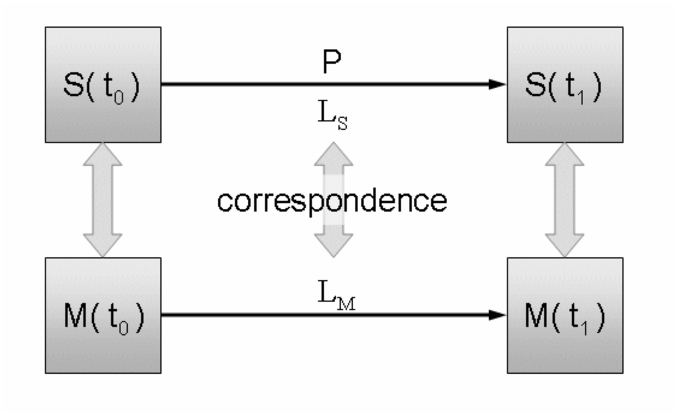


Figure 2—Correspondence between model behavior and system behavior

The role of this standard in developing system behavior models and in establishing the correspondence of such models with real or envisioned systems is addressed in Clause 5.

The representation of all cause-effect patterns of the system and a mechanism for inferring allowed evolutionary paths L_S is central to the reference model. These patterns are described as relationships between the interactions that occur through the shared phenomena P. These interactions are observables in E as they originate or terminate in the environment. The sequences and content of these interactions drive any particular evolutionary path. Clause 6 provides a more detailed description of the observables that have representations in M.

NOTE 3—The reference model in this standard is useful in practical ways, but it is not intended to be theoretically comprehensive. It is not required to be adequate for establishing global assertions about all possible behavior sequences.

Establishing a predictive capability requires that the behavior model must provide for representing the system at a particular time. It is an essential modeling problem to relate elements of the system *S* with elements in a model *M*. One purpose of the reference model definition is to identify and specify classes of elements in *S* and the corresponding classes of elements in *M* that represent them.

4.4 Test creation ability

Testing a system's behavior requires establishing known initial conditions $S(t_0)$, providing a specified sequence of stimuli and other inputs, observing the resulting system responses, and finally confirming that the system satisfies expected final conditions $S(t_1)$. This is a classical black-box viewpoint (with the advantage that it is applicable at multiple scales in extent and in time). A suite of such tests provides evidence that an implemented system realizes all the behaviors specified for meeting the stakeholder requirements. (These tests do not provide evidence that the stakeholder requirements themselves are satisfied by use of the system.)

Effective testing has two needs. The first need is for a surrogate to replace the environment *E* that is capable of generating stimulus interactions in *P* for *S* to use. The surrogate should also be capable of capturing response interactions in *P* that S produces. [In some cases, the surrogate might have additional capabilities for establishing the initial conditions $S(t_0)$ and for determining the final conditions $S(t_1)$.]

The second need is for test cases that specify appropriate initial conditions, sequences of stimuli to be applied, expected responses to be produced, and final conditions of the system. Based on the degree of refinement of a system behavior model, a sophisticated sampling strategy might by used to create an extensive set of such test cases. This approach to test case design is frequently called *model-based testing.* When the behavior model is part of the specification of the system to be realized, this may also be called *specification-based testing.*

The fundamental idea is to derive tests from the behavior model and to apply them against the realized system. There is no assumption about the structures and elements used to realize the system. The focus, rather, is to determine whether the realized system behaviors are equivalent with those behaviors intended for use in composing L_S, and ultimately, L_E. The tests are normally at the granularity of individual behaviors or small collections of individual behaviors. The test granularity should be commensurate with the controllability and observability properties of the system.

— *Controllability* refers to what the subject system can be directed to do by its stimuli (this might include control of its internal state as well as the control of its responses).

— *Observability* refers to what can be inferred about the subject system from observations of its responses.

Figure 3 illustrates this use of model-based test case specifications as system tests.

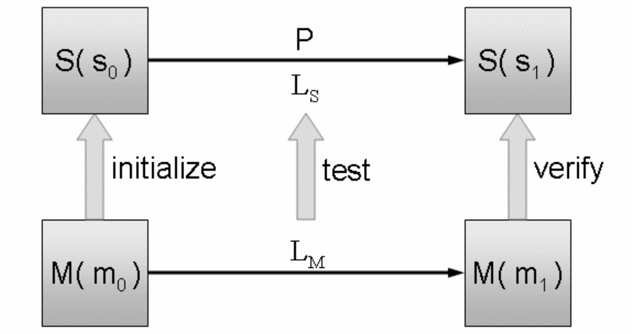


Figure 3—Model-based testing relationships

Initially, s_0 is the state of the system corresponding to the initial state of the model m_0. Similarly, s_1 is the final state of the system corresponding to the final state of the model m_1. The sequence of input and output interactions in L_S are defined in accord with those interaction sequences in L_M that move m_0 to m_1 in the model.

The use of the behavior model in this context depends explicitly on the predictive ability as required in 4.3. That ability implies that the behavior model contains all the information needed for the generation of representative sequences of individual behavior occurrences.

4.5 Life cycle roles

The declarative statements in the system behavior model *M* provide a "behavior specification" for the subject system, in that they provide a set of assertions that are to be true of the behaviors exhibited by that implementation.

The system behavior specification represents the earliest design decisions for a system, namely, decisions about how the system can interact with the application domain to bring into being the results described in the stakeholder requirements.

Prior to the implementation of a system *S*, that behavior specification can be analyzed for satisfaction of the stakeholder requirements in the expected usage contexts. As it identifies all behaviors to be supplied by system, it is the bridge between the stakeholder requirements and a technical solution. This specification might be validated against stakeholder requirements models and system usage models for the subject system.

During development, the system design can be analyzed to ensure that all specified behaviors can be produced and to understand the role of the various system design elements in producing them.

The system behavior specification also provides one basis for evaluating an implemented system that purports to comply with it. The behavior specification provides an explicit coverage target for final verification as it provides a list of behaviors expressed directly in testable (observable) terms.

— First, the specification can serve as a constructive test design basis for building a comprehensive suite of test situations, for parts, assemblies, and the complete system. The specification-based testing objective is to provide evidence of completeness and correctness over an arguable sampling of all application domain interaction sequences and behaviors. (In contrast, exploratory and context-based testing, following most probable error guidance, can be highly valuable during development for the detection of design defects, especially when product specifications are notional or incomplete.)

— Second, the model inference mechanisms can serve the development of expected system responses to those situations (test oracle). Subjecting the system to these tests and comparing its actual responses with the expected responses demonstrates explicitly the degree of compliance of the implemented system behavior with the intended system behavior.

The design process develops a physical and functional hierarchy, with specified interfaces between design elements. As at the system level, behavior models provide appropriate "behavior specifications" for parts as well as assemblies in the design, again providing assertions that are to be true of the behaviors exhibited by those design elements. These specifications comprise a portion of the specification tree that expresses the system design, and they provide a basis for behavior evaluation of realized parts and assemblies during system construction. (A more detailed discussion of the relevant process areas is found in Technical Report CMU/SEI-2002-TR-011 [B7].)

Following acceptance of parts, assemblies, and the system, these behavior specifications provide essential information to the collection of system reference and maintenance manuals.

4.6 Common interpretation

A behavior model that has an ambiguous interpretation with respect to a subject system is worse than no model at all. What is required is a common basis for model interpretation among people and tools that need to share an understanding of the subject system. Having such a "shared interpretation" means that each person or tool can make similar inferences about the subject system from the descriptive content of the model and the standard inference mechanism. Similarity between such inferences is the test for demonstrating a shared interpretation.

As described in 4.3, the reference model in this standard expresses the cause-effect patterns of system behavior as relationships between the interactions that can occur between the system and its environment through the shared phenomena *P*. Because these interactions are observables in the system's environment, they provide an unambiguous vocabulary of "physical" referents with which to interpret the model directly into observables. The descriptions of observables in the system behavior model include the following:

— Detailed descriptions of the controlling and observable interactions

— Identification of content and property relationship patterns among the controlling and observable interactions

— Causal and affective relationship patterns between controlling and observable interactions

Not all reference model concepts are directly observable. There are also *indirect observables.* An indirect observable may be called an *inferred* or a *derived* observable. An indirect observable is not knowable by actual observation, but could be determined from other direct observations. Inference without direct observation may be regarded as imputed reality. The most common imputed reality is called the *state* of the unit. Such imputations are accounting shorthand for the history of interaction occurrences that affect the unit's direct observables. Such bookkeeping practice localizes the effect of historically distributed interaction occurrences into a single current referent. (See 6.3.7 and 6.3.8.)

Clause 6 elaborates the classes of model observables and their basic characteristics.

5. System behavior metamodels

5.1 Developing the Conceptual and Data Metamodels for behavior modeling

An overview of the use of the system behavior reference model for the modeling of a subject system, real or envisioned, is illustrated in Figure 4.

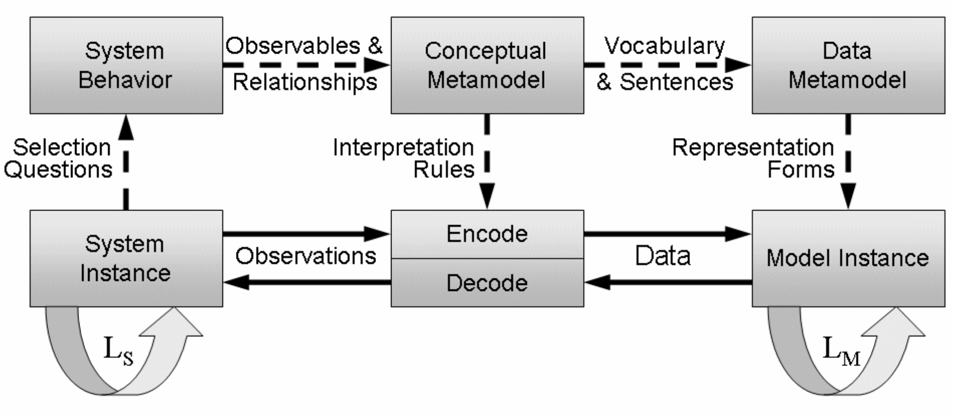


Figure 4—System behavior metamodels

The lower half of Figure 4 illustrates the development and use of a particular behavior model for a particular system. At the left is a particular system instance with its capabilities for behavior along its possible lifelines. At the right is a particular model instance that describes this system and its behavior. Connecting the model and the system are mapping links (Encode and Decode) that relate model data with actual system observations. Solid arrows indicate the specific relationships that exist there.

The upper half of Figure 4 illustrates the major parts of the reference model that inform the behavior modeling instance. The system behavior, Conceptual Metamodel, and Data Metamodel are meta-descriptions of the elements and relationships described in the previous paragraph. (Dashed arrows indicate relationships among the elements in the meta-description of model creation and use.) The system behavior box in the figure is a subset of all possible system characteristics that is selected for the purpose of answering specific questions of interest about system interaction with its environment. The Conceptual Metamodel provides a standard way of analyzing and characterizing those selected characteristics for understanding (or specifying) system behavior at some desired granularity of detail. The Data Metamodel provides a language for recording and storing a behavior description understood with the Conceptual Metamodel, and that description can be analyzed for inferring answers to the questions of interest about the system behavior. The model instance is the expression of the behavior model as words and sentences in the Data Metamodel language. The Conceptual Metamodel is the definition of the encoding and decoding relationships between the model instance and the system instance.

The purposes for which the behavior model is to be used determine which features of a system need to be included in the scope of the reference model. The system behavior includes a wide variety of behavior examples as seen in chemical, physical, thermal, electromechanical, software, manufacturing, and commerce system. It includes the rules of system evolution through its lifeline.

A Conceptual Metamodel is intended to provide a small number of unifying concepts and serves as a basis for explaining a model's capabilities. These concepts are abstractions of system behavior. The Conceptual Metamodel presents the domain of discourse for describing system behavior with a vocabulary for the observables of system behavior and for the relationships between these observables. It catalogs the type of facts to be used in their description. These elements are named, organized, and documented.

A Conceptual Metamodel provides the context for a common interpretation of a particular system behavior model in terms of the observables of behavior. To have a common interpretation means that two users of the model, whether these are two persons, two tools, or a person and a tool, can make similar inferences about behaviors of the subject system from the model's content. Similarity between these inferences is the test for semantic communication of models. This is what motivates the focus on observables as the basis for defining the semantic content of the behavior metamodel described in this standard.

In this standard, the Conceptual Metamodel for Behavior Specification uses the generic term *unit* to represent the particular object whose behavior is being modeled. A unit may be a system, a subsystem, a system element, an assembly of components, or a primitive component in a system.

The Conceptual Metamodel focuses on three aspects of behavior description:

— *BoundingIinterface:* Identifying the interactions between a unit and its environment, and describing relevant physical and temporal characteristics of those interactions.

— *Behavior Patterns:* Identifying the relationships between interactions and interaction histories that are made true by operation of the unit. Behavior also includes assumptions about the external world whose truth is prerequisite for correct unit operation.

— *Build Structures:* Identifying structures of interacting units in which unit behaviors are compounded through the coupling of interactions between units, thereby producing the behaviors of the assembly unit description.

While the Conceptual Metamodel identifies what things are to be said about the behavior of a system, it does not specify how to say those things. It is the Data Metamodel that provides an explicit specification of data elements, structures, and rules by which a particular system behavior model is built. It comprises a formal specification of the terms, the data, and the expressions with which a behavior model is expressed.

A Data Metamodel needs to provide representations for all observables, relationships, and fact types identified in the Conceptual Metamodel. It gives an explicit representation of the Conceptual Metamodel elements so that model facts can be recorded and manipulated. It provides a syntax for recording and communicating the contents of particular system behavior models in terms of defined data elements, data values, and expressions.

To implement the reference model as an engineering tool, a Data Metamodel would also need to implement numerous additional requirements that are not part of the Conceptual Metamodel. Those requirements include such implementation and usability concerns as model metadata, version management, and traceability, among others.

The purpose of this standard is to define a Conceptual Model for system behavior specification. General principles for the Conceptual Metamodel for behavior specification are described in Clause 6. The actual elements and relationships in the Conceptual Metamodel are designated in Clause 7.

5.2 Using the Conceptual and Data Metamodels for behavior modeling

Particular unit behavior models are instances of the Conceptual Metamodel for behavior specification expressed in accord with the Data Metamodel for behavior specification.

To build a particular behavior model for a particular system instance is an encoding process. The Conceptual Metamodel serves to assist the investigation, analysis, and understanding of the system's behavior, while the Data Metamodel provides a standard representation for recording that understanding.

Determining answers to questions about the features of the system's possible evolutionary lifelines E_S is accomplished by manipulating the standard representation of the model to instantiate a trace of the corresponding evolutionary lifeline $\mathcal{E}_M$. The Conceptual Metamodel needs to provide the appropriate conceptual framework for making such inferences. The Data Metamodel needs to provide the corresponding elements, data structures, and algorithms for executing such inferences.

To interpret a modeled behavior result into a statement of expected behavior by the particular system instance is a decoding process. Again, the Conceptual Metamodel guides the interpretation of inference data obtained from the particular model for the observable elements and relations those data imply about the particular system instance.

6. Conceptual Metamodel foundations

A model is a mental construct that represents some aspect of the modeled object. Models are constructed to enable reasoning within an idealized logical framework about the object. Models are idealized because they omit certain aspects of the modeled object or they make explicit assumptions that are expected to be true. The effect of omissions and assumptions is to simplify the use of the model as an inference tool while not significantly damaging the inferences made from it. The purpose of this clause is to identify and describe the worldview used in the Conceptual Metamodel concerning the observables of unit behavior.

The Conceptual Metamodel of unit behavior uses a worldview elaborated from three general principles:

- Units are the agents of all behavior in the model.
- Interactions are the percepts of unit behavior.
- Causal and affective patterns among interactions are the concepts of unit behavior.

Complementing the elaborations of these principles is an inference concept for predicting observable unit behavior and a composition structure for relating behaviors of an assembly to behaviors of the units composing that assembly.

Throughout this and subsequent clauses, concepts are illustrated through textual and graphical examples. Generally, units in these illustrations have names beginning with the letter “U,” ports have names beginning with “P,” and interactions have names beginning with “J.” The notation *U1.P2* references port *P2* on the boundary of unit *U1*. An interaction *J3* at that port is denoted by *U1.P2>J3* if it is an output and *U1.P2<J3* if it is an input.

6.1 Agent of behavior—Unit

Unit is the Conceptual Metamodel element that represents a system, a subsystem, a system element, an assembly of components, or a primitive component in a system. A unit of the kind addressed by this standard has a physical realization and exhibits repeatable, or at least predictable, behaviors that are used to bring about intended effects in their environment. To have predictable behavior requires that interactions with the unit occur in expected patterns. Having knowledge of the unit’s history of interactions (the current point on its lifeline) allows a prediction of its responses to a particular stimulus. Even when a certain randomness is part of a unit’s characterization, there is some specified domain in which the random feature operates, and the result must be in that domain with an equi-probable result occurrence across an ensemble of behavior instances.

To have repeatable behavior requires particular capabilities of both the unit and its environment. It requires the ability to reset the unit to a particular point on its lifeline, to reestablish the previous operational environment (at least so far as the unit’s ability to sense that environment), and then to stimulate it exactly as done previously.

NOTE—For some units, repeatability of behavior with the realized unit may be difficult, impractical, or impossible to demonstrate, but that does not invalidate the predictability of the behavior.

The unit is the agent of all behavior in the Conceptual Metamodel. It is the subject for which behaviors are specified. All statements in a single behavior model ultimately provide facts about it.

In the Conceptual Metamodel, a type of unit is defined by the collection of interfaces it shares with its environment and the itemization of the distinct interactions that can occur at those interfaces. The definition of the unit lists all interfaces and interactions the unit can have. These definitions provide a fixed basis for specifying the behaviors of the unit.

The reference model thus assumes a stable and time-invariant unit definition. There is no notion of dynamic unit definition. A unit does not have one set of possible interfaces at one time and a different set of possible interfaces at a different time. There is no provision for the "creation" or "destruction" of interface definitions as a result of unit behavior. The reference model does have the notion that interfaces (ports) might be active at one time and not at another time (described at the end of 6.1.3).

Having stable existence means that instances of units and members of the bounding interface set can be distinguished from other instances of their kind, that all instances exist concurrently, and that they can be assumed to be always present. When their instances are described in a behavior model, there is no time of existence associated with them because their presence and description are valid for all times.

Specifically, the stable elements of unit definition are the ports in its boundary and the various kinds of interactions that can occur through its ports.

6.1.1 Unit boundary

As commonly conceived, a boundary is a physical or conceptual surface that encloses something of interest. Questions about the nature and the reality of the boundary are usually complex and should be addressed at the outset of behavior model development. An unambiguous specification of boundary might not always be obvious. Clearly defining the behavioral boundary of a unit is the foundation for developing a useful behavior model.

Outside the unit boundary is the application domain, an environment of existing things and relationships in terms of which the stakeholder requirements for the unit are defined. It is to these requirements that the subject unit delivers value through its causal and affective behaviors that interact with unit(s) in that environment. Numerous initial decisions about these interactions need to be made before behaviors can be identified. Among these decisions are the following:

- With which units in the environment should the system interact for sensing and affecting the application domain?
- What phenomena do the system and the selected environment units have in common that can be used for interacting?
- What interactions can be effectively exchanged by between system and environment?

The behavior boundary between the system and its environment is the edge at which interactions between the system and the environment can be observed. That boundary is represented in the behavior model by the identification and characterization of those interactions. Potential interactions that are not explicitly included in the model are assumed to be nonexistent or inconsequential.

The system unit has an existence independent of the environment units. Units are distinguishable; each as its own identity. It owns an implementation for creating its behavior. Operations within the system are independent of operations within the environment, except when interactions through shared phenomena allow specifically defined influences of one unit's operation on another unit's operation.

In physical realization, boundaries segregate matter, energy, information, and mechanism into what "belongs" to each independent unit. The boundary allows certain interactions, and it prevents other things in the system and environment units from affecting one another.

Boundary identification differs somewhat in design development. The interaction and behavior commitments at the system level are fixed, but the interfaces between the independent subunits in the assembly are free. That freedom can be exploited by choosing unit-to-unit interfaces to optimize other properties of the system (e.g., reliability, safety, or testability), or to simplify implementation (e.g., adapt commercial units, reuse existing units, or facilitate producibility).

Boundaries are two-sided. Interacting units share a part of their individual boundaries with each other. The collection of all pair-wise boundaries around a particular unit constitutes the unit's boundary. The unit is an aggregation of its characteristics and features enclosed by its boundary.

The specification of all interaction occurrences defines a unit's behavior boundary and provides the vocabulary for expressing all behavioral traits of the unit.

Figure 5(a) shows two interacting units with their shared boundary segment allowing a specific kind of interaction by which unit *E* can influence unit *S*. Identifying and characterizing this interaction defines the boundary between *E* and *S*. Each unit is assumed to have independent existence and operation. Unit *S* can be conceptually separated from unit *E* as shown in Figure 5(b) without altering the behavior of either unit. The interpretation of the interaction in Figure 5(b) is the same as in Figure 5(a)—it is an identity between what is at the boundary of *E* and what is at the boundary of *S*. It is a common topological fiction to assign unique names to the interaction produced by *E* (for example J_E) and the interaction accepted by *S* (for example J_S) for the purpose of independent specification and development of the two units. When considering the combined behaviors of *E* and *S*, this identity of J_E with J_S would need to be recognized. (It would be possible for J_E and J_S actually to be different so long as they obey certain subsumption criteria. See 6.5.1.)

When poorly specified boundaries for unit *E* and unit *S* give slightly different views of the *E/S* interface, it could happen that J_E and J_S are not identical. It would be simplistic and counterproductive to admit some corruption or lossiness between the boundary of *E* and the boundary of *S* without identifying an agent accountable for this. In such a situation, Figure 5(c) might be a more accurate representation of the situation. A third unit *X* is introduced to represent the transmission or transport channel having the lossy behavioral characteristic. Between *E* and *X* there is identity of interactions and between *X* and *S* there is identity of interactions. In Figure 5(c) it is again true that the interacting units *E*, *X*, and *S* may be conceptually separated without altering the behavior of any unit.

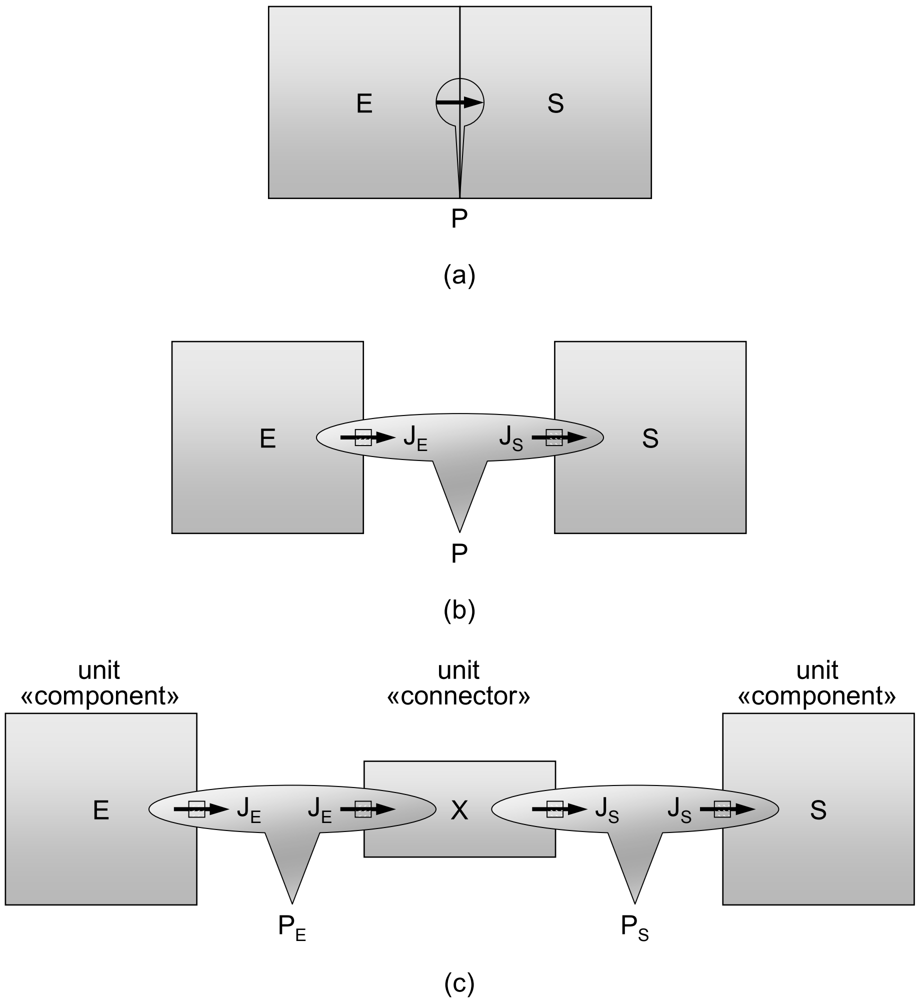


Figure 5—Units as agents of behavior

6.1.2 Unit stereotypes

The pattern illustrated in Figure 5(c) shows three units and two stereotypical roles: «component» and «connector». These roles are purposive, with the «component» designation suggesting some kind of transformative relationship between inputs and outputs, and «connector» designation suggesting an identity of content between input and output, but output at a different place and time than input. It is also possible that the phenomenological interfaces P_E and P_S are entirely different, as would be the case, for example, for a "smart" sensor that detects thermal emission from *E* and varies a resistance in *S*.

Frequently, component units in a system are application-specific, providing the specific functional transformations that offer value into the application domain. As frequently, the connector units in a system are generic, though no less important overall to the successful operation of the system.

Recognizing a distinction between component and connector units is a useful approach to the identification of specification units. In particular, the explicit identification of connector units has several advantages. Connector units provide a distinct locus for capturing various allocated communication and coordination behaviors.

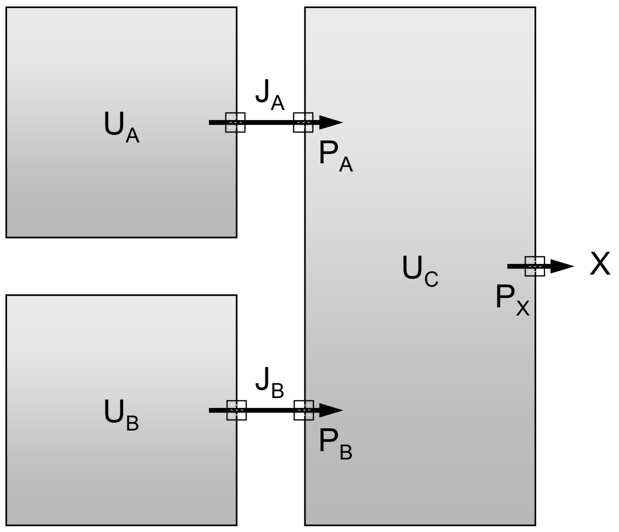
- Specific patterns associated with point-to-point, multicast, store-and-forward behaviors are directly specifiable for a connector unit.
- Deposit and removal behaviors for buffering and delay units (e.g., data memory, electrical capacitors, bins for materials, tanks for liquids) can include polling, contention mediation, and update synchronization patterns.
- Non-ideal, uncontrolled features such as incidental noise, distortion, corruption, spoilage, loss, or other forms of interaction degradation can be incorporated into the net behavior of a composition of individual units.

Whereas the «component» and «connector» stereotypes are valuable guides for design partitioning, any actual unit could have both transformative and coordinative behaviors included in its specification.

6.1.3 Unit ports

When unit *S* is conceptually separated from unit *E*, the boundary of each, as shown in Figure 5(b), is annotated with a small square to indicate the interface that exists between them. This shared interface is called a *port* and appears twice, once in *S* and once in *E*, as each unit sees that shared interface from its own perspective. The two ports thus created are called *conjugate ports* to emphasize this identity from opposite perspectives. While both ports see the same interaction, one labels it J_E and sees it as an output while the other labels it J_S and sees it as an input. Conjugate ports have the same interactions but specified with opposite directions.

For accountability in isolating a single unit from those with which it interacts, it is necessary to identify each of its various phenomenological interfaces uniquely. In Figure 6, for example, unit *C* could experience two input interactions J_A and J_B from two different units U_A and U_B. Each of these interactions is of the same kind *J*, for example, but different behaviors could result from the timing, duration, or content of each interaction. These interactions come through different shared interfaces, both an U_A–U_C interface and a U_B–U_C interface, so each shared interface is given a different label to identify it. These labeled interfaces in the unit boundary are called ports. They are labeled as P_A and P_B in the figure and represent the distinct parts of U_C's boundary that are pervious to *J*.



J_A and J_B are both of interaction kind J

Figure 6—Ports as shared interfaces in boundaries

As interactions between the unit and its environment occur through shared phenomena (see 6.2), so the realization of interface ports is through elements that sense, transport, or produce those phenomena. Real ports have physical features with an identifiable passive or active realization. The notion of shared phenomena extends beyond simple physical basis to include some meaning of the interaction with respect to observable behavior. For example, when a communication port impresses patterns onto electromagnetic signals, some signal patterns have different structures and different interpretations than others. In a model, different signal "types" may be designated for the possibilities of each structure. The model interface then associates several kinds of interactions with a single port.

The designations by which boundary correspondence is made between the behavior model and the real unit (already existing or to be created) are important. Ambiguity about the actual boundary elements (ports and interactions), in terms of which the behavior model is expressed, can introduce significant errors into the model. The model is a symbolic representation of the system's reality, and clarity in symbolic reference reduces the likelihood of misinterpretation in the model.

Ports can be bidirectional or unidirectional, allowing interactions to occur in either direction or in one direction. Some interactions might be identified as allowed inputs and other interactions might be identified as allowed outputs.

Ports serialize interactions in a particular direction. An interaction might have a structure of concurrent content, but during its occurrence, another independent interaction cannot also occur in the same direction at the same port. If independent interactions can occur concurrently at the boundary of a unit, then multiple ports are used in a specification to distinguish all interaction situations that might occur. Figure 6 illustrates this possibility with the input ports P_A and P_B for unit U_C.

A port, as an engineered interface, might allow interactions at one time and disallow them at others. That is, a port might be able to control its availability to interaction occurrences. (Interaction availability is discussed in detail in 6.3.2.) Availability control can allow the occurrence of some interactions defined for the port and can inhibit the occurrence of other interactions defined there. Thus, the ports in a unit's boundary provide a controllable exposure of the unit to the behaviorally meaningful interactions with the environment, subject to the unit's own behavior.

NOTE—Interactions that are not behaviorally meaningful are those that would damage, disable, or destroy the subject unit (e.g., crushing, exploding, melting, or fragmenting). These differ distinctly from behaviorally meaningful interactions that may trigger the unit to perform intentionally self-destructive actions.

6.2 Percepts of unit behavior—Boundary interfaces

Interactions are the percepts (i.e., the primitive observables) of behavior. The direct observables associated with an interaction are transitory. Such characteristics are momentary and temporally-bounded in existence because interactions are momentary and temporally bounded in existence.

Having a temporally bounded existence means occurrences of these observables are not distinguishable from other occurrences of their kind without placing them into a temporally distinguishing context. That is, the occurrence that is happening "now" is temporally distinguished both from the one that happened "before" and from the one that will happen "after," which is not yet known. Therefore, when such occurrences are described in a behavior model, a (relative or absolute) time of existence is associated with them. When the model is interpreted along a timeline, an occurrence has only one interval during which it exists.

The transitory observables of an interaction occurrence are its begin and end events, its properties at the boundary port, and the domain of future effects associated with its occurrence.

6.2.1 Interaction identities

From the perspective described in 6.1, interactions are not objects. They are identities (with a particular phenomenological basis) that occur at the interface between two units. Certain characteristics of that identity are created by one unit, the interaction "source," and are adopted by the other unit, the interaction "destination." This is the meaning of *shared interface.*

Deviation of notion and language from this perspective loses clarity and accountability in the understanding of the system and the accuracy of its specification. Interactions can become things granted an independent existence, as things to be "sent" and "received." Too often, there is thought to be an interaction generated by the source at one time and place that is received at a later time and in a different place at the destination. It becomes an assumption in each unit that "what is sent is what is received," with no accountability for the truth of that statement. The notion of interaction as identity guarantees that source and destination are not materially distinguished in place or time.

NOTE—Two units with a shared interface whose two sides are materially separated from one another require complementing by a third unit [as in Figure 5(c)] whose behavior is to enable the transfer of interaction identify from the one unit, at one place and time, to the other, at another place and a later time. Nowhere is there an unaccountable existence of interaction after "leaving the source" and before "arriving at the destination."

As a fundamental principle of empiricism, all a unit can "know" about the objects in its environment is known through the interactions it shares with them. Likewise, a unit is known to the objects in its environment only through the interactions they share with it.

Interaction occurrences are those finite time intervals in which interaction identities are passed from source to destination.

6.2.2 Interaction characteristics

Interactions have the effect of altering the lifelines of the source and destination units from what they would have been in the absence of interaction. In the Conceptual Metamodel of system behavior, interactions are explicitly characterized. Interaction identities are observable—they can be sensed and measured by independent observers at the interface between units. This property gives interactions their objective basis in reality.

Interaction identities have an existence that is momentary in comparison with other things (e.g., ports) whose existence is durative in the unit. An interaction has a begin time at which it is perceptible in the shared phenomena at the port in the unit boundary. It has an end time after which it is no longer perceptible. If the granularity of time in interaction perception is larger than the difference in end and begin time, an interaction may be perceived as occurring at a "point" in time, so it is called a *point interaction.* If the granularity of time is small, an interaction may be perceived as occurring through an "interval" in time so it is called an *extended interaction.* Some behavior properties might be easier to describe with large time granularity, others might be easier to describe with small time granularity. Knowing the time granularity used in specifying a particular behavior pattern is necessary for an unambiguous interpretation of the intention.

The contents of an interaction identity are those characteristics that are created and sensed at the interface. Generally, interaction content may be categorized as energy, matter, or information. Energy and matter differ from information in that they are normally subject to conservation laws; conservation laws are meaningless for information. Units may be regarded as having an energy or matter content that can be changed by interaction with other units. The interaction identities may state equal and opposite amount changes (lump) for the interacting units, or equal and opposite rates of change (flow), with each unit experiencing a single begin time and a single end time. Matter differs from energy as it has mass and volume considerations.

By its fundamental nature, information is not conservative. Information is a patterned deviation from randomness that is embedded in an energy or matter medium. An information interaction is one that communicates information through the creation and sensing of the particular pattern in the host medium shared by the two units.

To allow certain details of behavior to be represented in behavior specifications, it may be useful to represent an interaction as an aggregate of other interactions. These other interactions are component interactions of the aggregate. Aggregation allows the recognition and identification of unit responses that are accountable to an occurrence of the aggregate interaction as well as unit responses that are accountable to the occurrence of the individual component interactions.

Some common examples of aggregate interactions include the following:

- In a solid material interaction, an aggregate interaction of "shipment" may be accounted for its occurrence, whereas the "items" within the shipment may be accounted as an inventory count. In some cases, units may need to be distinguished by order of processing. These transfers occur sequentially.
- In a flow of material and energy, both mass and heat are transferred across a port. Different mass and heat may be put into the interaction occurrence by the interaction source, and these can be separately determined and used by the destination unit.
- In a flow of information, separate information streams can be multiplexed into a radio signal by the signal source. At the antenna port, there is a complex signal transferred to the destination unit. The signal can be demultiplexed by the unit to retrieve the separate signals.

Aggregate interactions at a port are not artificial constructs, but are a description of physical constructs. Their identification and definition are subject to several constraints.

- An instance of an aggregate interaction is identified at a port with a direction of influence.
- An occurrence of an aggregate interaction is observable at the port.
- Multiple occurrences of an aggregate interaction are serialized by the port realization.

Similar statements apply to component interactions in the aggregate.

- An instance of a component interaction is identified at the same port with the same direction of influence.
- An occurrence of a component interaction is observable at the port.
- Multiple occurrences of a component interaction are serialized by the port realization.

These observable characteristics of aggregate and component interactions allow two kinds of composition: sequential and concurrent.

Sequential interaction components have begin and end times that occur between the begin time and the end time of the aggregate interaction. They do not have overlapping intervals of occurrence. They are already serialized in the aggregate structure. The observability of sequential component occurrences also provides a greater granularity in temporal synchronization between the unit and its environment. Examples of sequential aggregate interactions include byte transfer on a serial line, time-division signal multiplexing, and continuous flows of material signals with time-dependent properties.

Concurrent interaction components have the same begin and end times as the aggregate interaction. The components can be compounded (source) and fragmented (destination) without loss of content, i.e., components retain identity on both sides of the port. In the aggregate interaction, however, the component occurrences are physically integrated in the aggregate interaction so occurrence of the aggregate interaction at the port means occurrence of every component at the port. Examples of concurrent aggregate interactions include byte transfer on a parallel line, frequency-division signal multiplexing, and non-reacting chemical mixtures.

Simple interactions therefore have simple content expression as a collection of properties, while aggregate interactions have a structured content expression as a sequential or concurrent composition of other interactions. This structuring may also be repeated as an aggregate interaction of aggregate interactions.

6.2.3 Interaction observability

While interactions always have begin and end times of occurrence, those particular times are not necessarily important for behavior specification. Similarly, while interactions always have a content that is distinguishable from the observable conditions at the port before and after the interaction begin and end time (so that the begin and end time points are themselves discernible), that content is not always important to the behavior specification.

The Conceptual Metamodel distills these characteristics into the observables of *event* and *property*. To these is added the observable of *effect*. The model of an interaction need not describe all three of these characteristics. Interactions of significance for the behavior model are those exhibiting one of the following three combinations of observables:

- Effect and event
- Effect and property
- Effect and event and property

Resulting effect is the first fundamental notion of the observable interaction. The observation of effect gives modeling significance to an interaction. An input interaction that has no observable effect on the unit is irrelevant in a behavior model of that unit. If, because of the occurrence of a particular input interaction, some later output interaction is modified from what it would have been, then that input interaction has had a resulting effect. Similarly, if some output interaction causes a change in an external unit receiving the interaction, that output interaction has had a resulting effect.

Event occurrence is the second fundamental notion of the observable interaction. There is a moment at which some perceptible phenomenological change (energy, matter, or information) occurs at the port of the unit. The singular time at which perceptible change occurs is called an *event.* Events are points in time marked by a perceptible change. For an interaction, the initial change from absent to present signals a "begin" event while a final change from present to absent signals an "end" event. The interaction event is significant if the purpose of the interaction is to initiate or terminate a behavior of the unit.

Often, a behavior of a subject unit is initiated by a particular interaction caused by the environment; this interaction is called the *stimulus*. There are also one or more interactions caused by the subject unit; the collection of these interactions is called the *response*.

Either the begin time or end time (or both) at which a stimulus interaction occurs are noteworthy events. There might be some response interactions whose begin and/or end events are expected to occur with some particular temporal relationship to a stimulus initiation event or another response event; these are also noteworthy events. The begin event of the stimulus and the end event of the last response are sometimes used to measure of the total time for a particular behavior.

Property transfer is the third fundamental notion of the observable interaction. A property is a measurable phenomenological characteristic of an interaction at the port of a unit. For a stimulus interaction, it is often the case that there is not particular meaning to the measurement of any interaction property; rather, it is simply the detection of a particular property change that constitutes an initiation event and the property itself has no effect on the unit behavior.

For most interactions with most systems, however, property transfer is the whole purpose of the interaction. For matter and energy transfer, conservation requires equal and opposite changes of matter or energy content in the interacting units. For information transfer, pattern transfer is a duplicative process in the host medium. The selection, denomination, and representation of properties in unit interactions are modeling arts. Possibilities are multiplied through actual and conceptual structuring of simple properties into complex properties (e.g., consider the description of a Gregorian date in terms of year, month, and day, or the description of a gasoline flow in terms of temperature, viscosity, chemical composition).

Not all possible measurable properties have significance in a behavior model. For example, if a series of voltage pulses on a communication cable is used to duplicate a bit pattern from a communication link to a computer input port, there is negligible change in the electrical energy content of the computer and of the communication interaction. Only the bit pattern property is relevant to the behavior model. Conversely, for some behavior models, it might be important to include mention of a power supply input interaction in whose absence unit behaviors will not be exhibited.

6.2.4 Interaction instances and occurrences

A model of observed behaviors of the unit would contain declarations about interaction occurrences that are tagged with actual times when those occurrences are happening. A model of observed behavior does not provide a predictive ability, simply an historical log. To be predictive, actual interaction occurrences must be abstracted to a declaration of possible interaction occurrences.

For these transitory observables, the Conceptual Metamodel distinguishes between *occurrences* of interactions and *instances* of interactions. Refer to the unit labeled U_C in Figure 6 for an illustration of this distinction. In that figure, the shared input interaction from unit U_A is labeled with a J_A, the one from unit U_B is labeled with J_B. Though these two interactions are of the same kind, this does not imply their occurrences to be identical in content or in time of occurrence. The Conceptual Metamodel regards the J_A at P_A and the J_B at P_B as two distinct *instances* of possible interactions of the same kind J. The symbols J_A and J_B each represent the same set of interaction possibilities named J.

When one of the possibilities in J actually occurs, with some begin and end time and content at port P_A, for example, that is an *occurrence* of the instance J_A. Actual occurrences of interactions J_A and J_B are completely independent from one another. The definition of the interaction possibilities in J is actually the definition of an interaction type. Types are intensive definitions that can be interpreted and expanded into an extensive set enumeration of all interaction possibilities.

The Conceptual Metamodel provides for the specification of interaction types and interaction instances. Interaction occurrences are not part of the metamodel. Rather, they are part of the predictive inference mechanism by which dynamic behavior of the unit is generated from the behavior model.

6.2.5 Interaction observations

With respect to the behavior metamodel, engineered systems differ from natural systems in a special way, and for an important reason. Generally, the causal and affective behaviors of natural systems have relatively repeatable patterns over very large ranges of interaction phenomena. At extremes of dimension and energy, familiar patterns are replaced with less familiar patterns, but these changes are themselves relatively repeatable meta-patterns. And so goes natural philosophy, building and validating models of these behaviors and meta-behaviors to understand better the natural world.

However, the engineered systems addressed in this standard are to be conceived and developed to serve particular human purposes in defined contexts. In those contexts, they are not open to all possible interaction phenomena. For those purposes they are bounded in dimension and energy. The specifications of intended causal and affective behaviors of these systems need refinement to address not only multiply partitioned domains of interaction occurrence content and sequence, but also intended discontinuities in the patterns of behavior to be exhibited for those different partitions.

The metamodel uses conditions to express and identify these points of behavior discontinuities with respect to interaction properties and time.

A condition identifies one or more model elements and states a predicate about them. This predicate may be a statement about some characteristic of a model element, a statement about a relationship between model elements, or a logical combination of such statements. Inherent in such a statement is the notion that it references characteristics or relationships that can be observed or determined at a point in time, and the statement is then evaluated as TRUE or FALSE. How the implied observations are made and how the truth or falsity is evaluated is not prescribed. That is, the Conceptual Metamodel does not require a particular sorted Nth-order logic for expression (although a Data Metamodel may incorporate some restrictions). Conditions are simply regarded as functions having a binary result at a point in time.

With respect to interactions, the *condition expression* is a predicate stated in terms of interaction instances, with the instance name standing for one member of the collection of possible interactions that could occur. The *condition value* depends on the particular instance member that occurs, i.e., an occurrence.

Consider two simply unary conditions on a sensor circuit producing a time-dependent analog voltage input to some system during some monitoring period. This is an interaction with time-varying content, as the voltage varies during the monitoring period.

Example 1: Property condition

While the voltage V varies, it is never zero for a functional sensor. An occurrence of this interaction between its begin and end times t_0 and $t_1$can be characterized as a "valid" or an "invalid" possibility depending on whether the following condition on content V is TRUE or FALSE:

Condition V_Valid: $V(t) > 0 \quad t_0 \leq t \leq t_1$

Clearly, this describes some observation to be made (values of V during the monitoring interval) and a predicate that can be evaluated from those observations.

This condition serves to partition all occurrences of the sensor input interaction into two subsets, those that are "valid" and those that are "invalid." This is a valuable refinement in the specification as it enables a distinctly different behavior to be specified for "valid" sensor input than for "invalid" sensor input. For example, the occurrence of an invalid monitoring interaction might be announced as an equipment failure.

Example 2: Event condition

As the voltage varies during a monitoring interval, it might be above or below some prespecified reference value V_0. As in Example 1, there might be distinct behaviors to be specified for each case. The required observations and comparison are expressed in the following condition:

Condition V_Over(t): $V(t) > V_0$

As the sensor voltage is time-varying, this is a time-dependent condition. Between the begin and end events of the interaction, this condition might change from TRUE to FALSE and from FALSE to TRUE some number of times. A separate behavior may be specified for those periods when the condition is TRUE than is specified for those periods when the condition is FALSE. Furthermore, this condition can serve to define a marking of points in time, specifically, the time points at which the value TRUE is replaced by FALSE and vice versa. This defines a sequence of observable events that could be significant for anchoring various temporal characteristics of the behavior against time in the application domain.

6.3 Concepts of unit behavior—Behavior patterns

Whereas interaction occurrences are the sensible percepts of behavior, the abstracted patterns of causal and affective relationships among these occurrences are the concepts of behavior.

The observables associated with the behavior concept are inductive abstractions. These are the discernible patterns among the occurrences of transitory observables, as well as deterministic structures of such patterns. Being discernible patterns means instances of these observables are recurring sequences of input and output interactions that exhibit causal or affective relationships. These relationships are ones required to be true of an implementation of the subject unit. They are also relationships that might be determined or demonstrated experimentally (to some degree of certainty) as tests of the implementation. Controllability and observability properties of the subject unit are critical to the development of substantive evidence of acceptable implementation.

6.3.1 Behavior and causality

Behavior and causality are complementary views of the relationships among interactions and units.

— Behavior centers on a unit and the interactions between it and its environment. It is concerned with correlating the response a unit produces to the stimulus that causes it to do so. The core issues concern the identification of sequences of stimuli and inputs that initiate and feed the behaviors of the unit, and the identification of corresponding responses and outputs that deliver the intended effects of those behaviors into the unit's environment. Behavior asks, "What can a unit do?"

— Causality centers on an interaction and the units it relates. It is concerned with allocating controls over each aspect of the interaction. The core issues concern the identification of control over its occurrence event, control over all properties of its occurrence content, and control over its rate of occurrence. Causality asks, "Who is in charge?"

Behavior relationships give definition and dynamics to functional dependency. Causality tracks the control of coordination through interactions among units. Dynamic behavior modeling includes both functional dependency and control coordination.

Causality requires an explicit consideration of interaction coordination. For every interaction that occurs between a unit and its environment, either the subject unit or an external unit in the environment is responsible for its actual occurrence. It is not possible for the source and destination unit to control simultaneously the same interaction occurrence.

Thus, in the specification of a single unit, the *occurrence* of every specified interaction at every unit port is either controlled by the subject unit and allowed by the external unit on the other side of the interface, or it is controlled by the external unit and allowed by the subject unit. This control includes the timing, the frequency, and the duration of the interaction (i.e., the begin event and end event times). If the unit not in control is unable to accept the full range of interaction frequency and durations that could occur at its ports, then dependable behavior of the unit requires the design to establish coordination agreements between the unit and its environment.

Though the *occurrence* of an interaction is controlled by either the source or the destination, causality requires that the *properties* of interactions can be controlled only by the source of the interaction. This requires explicit consideration of property domain coordination. The property values for every interaction occurrence are constrained at its source by natural or artificial limits. If the destination unit is unable to accept the full range of property values that could be produced at its ports, then dependable behavior of the unit also requires design to establish coordination agreements between the unit and its environment .

The *rate* at which interactions can transfer material, energy, or information depends on the physical characteristics of the interactions and comparative motive forces in the source and destination units.

Depending on the time granularity assumed in the description, rates might vary from very small (finite amount over a long interaction duration) to very large (a batch quantity within one grain of elapsed time). The reference model allows interactions to have either a rate or a net amount property (either one with some domain of occurrence values) and duration (depending on the time grain of description).

6.3.2 External stimulus and response

It is frequently true that input interactions are stimuli and output interactions are responses, though this is not always true. "Direction" is a simple binary characteristic of an interaction and the adjectives "input" and "output" describe the direction of an interaction relative to a subject unit. Inputs are identities of material, energy, or information transfer that are established by an external unit as the interaction source and matched by the subject unit as the interaction destination. Outputs are identities established by the subject unit and matched by the external unit.

There are three different aspects of control on an interaction to be allocated between a subject unit interacting with an external unit.

— Interaction occurrence event: Which unit controls this?

— Interaction properties: Which unit controls this?

— Interaction rate: Which unit controls this?

The principal issue for causality is the control of interaction occurrence, bringing an interaction into existence. Interaction properties and interaction rate are characteristics of an existing interaction occurrence.

A *stimulus interaction* is an input interaction or an output interaction whose occurrence event is controlled by an external unit and affects the subject unit. The interaction is created by an external unit and, at least sometimes, is allowed by the subject unit; otherwise the subject unit could not be affected. The distinguishing feature of a stimulus is that its occurrence is not controlled by the subject unit. The occurrence of a stimulus synchronizes the lifeline of the subject unit to the lifeline of the external unit.

A *response interaction* is an input interaction or an output interaction whose occurrence event is controlled by the subject unit and affects an external unit. The interaction is created by the subject unit and, at least sometimes, is allowed by the external unit; otherwise the external unit could not be affected. The distinguishing feature of a response is that it is not controlled by the external unit. Its occurrence synchronizes the lifeline of the external unit to the lifeline of the subject unit.

NOTE 1—It is important not to equate the notion of control of an interaction occurrence with the notions of source, destination, and direction of the interaction. The word *stimulus* is not synonymous with *input.* The word *response* is not synonymous with *output.*

Although the stimulus presented to the subject unit could be an input *to* that unit or an output *from* that unit, it is proper to say "a stimulus *to* the unit" because the stimulus originates outside the unit and precedes the unit behavior. Similarly, whether the response generated by the subject unit is an output *from* that unit or an input *to* that unit, it can be said to be "a response *from* the unit" because that response originates within the unit and as a result of the unit's behavior.

For all interactions, regardless of which unit controls its occurrence, it is the interaction source unit that controls the properties of interaction content.

For two units to share an interaction of material, energy, or information, the interaction control by one unit must be matched by an interaction allowance by the other. The unit allowing the interaction makes it *available* to the controlling unit.

NOTE 2—For example, an input to the subject unit might be controlled by an external unit and allowed by the subject unit. Alternatively, an input might be taken under control of the subject unit while the external unit allows this. A similar statement holds for output from the subject unit.

This gives the following four cases. (Interaction directions are stated with respect to the subject unit.)

— The subject unit makes available an input interaction that allows an external unit to produce that input to the subject unit when desired. This *externally* controlled "push" interaction would be a stimulus to the subject unit.

— The subject unit makes available an output interaction that allows an external unit to get that output from the subject when desired. This *externally* controlled "pull" interaction would be a stimulus to the subject unit.

— The external unit makes available an input interaction that allows the subject unit to get that input when desired. This *internally* controlled "pull" interaction would be part of a response of the subject unit.

— The external unit makes available an output interaction that allows the subject unit to produce that output when desired. This *internally* controlled "push" interaction would be part of a response of the subject unit.

The pattern in these cases is apparent in the following table:

Case	Direction	Interaction controlled by	Interaction allowed by	Role in behavior
1	In	External unit, push	Subject unit	Stimulus
2	Out	External unit, pull	Subject unit	Stimulus
3	In	Subject unit, pull	External unit	Response
4	Out	Subject unit, push	External unit	Response

The unit that offers an available interaction is an enabler. It makes an interaction available to the controlling unit to cause, and the controlling unit is free to cause it or not. The offering unit might exert an *availability control* over the interaction, because it determines whether or not to make the interaction available. When the interaction is available, the controlling unit has *occurrence control* over its actual occurrence. Availability control by the offering unit cannot cause an interaction occurrence; it can only prevent its occurrence.

NOTE 3—As an example, consider a cooling pond (source unit) that a manufacturing process unit can use. The process unit has occurrence control here. However, the pond might be too low in late summer, denying availability so the process unit cannot use it then.

In summary, a stimulus interaction (controlled by the external unit), by definition, matches an available interaction allowed by the subject unit. A response interaction (controlled by the subject unit), by definition, matches an available interaction allowed by the external unit. The occurrence of stimulus and response interactions is uncoordinated between the units and unpredictable by the unit not in control.

When a unit response includes generating an output for initiating an external unit to provide input to the subject unit, the subject unit is imposing an *obligation* on its environment. Whoever or whatever uses the subject unit for delivering an intended effect is obligated to ensure that the necessary input is available from the external unit. Failure to meet that obligation prevents the subject unit from delivering its intended behavior.

In contrast to the uncoordinated interactions of stimulus and response, there are other interactions whose occurrence is coordinated between the source and destination units and is anticipated by the unit not in control. *Coordinated interactions* involve intent, so they do not appear in natural physical/thermal/chemical systems. They are widely present in biological and engineered systems, however. Coordinated interactions comprise a pair of interaction occurrences in opposite directions, serving as a request and a reply. They arise when a subject unit, in the process of developing a response to some stimulus, engages an external unit's services for its own purpose. From the subject unit perspective, the other unit is regarded as a subordinate. The subject unit generates an interaction (the "request") that is a stimulus to the external unit. It does this in anticipation that the external unit will then generate a response (the "reply") that benefits the subject unit.

NOTE 4—Examples of such subordinate engagements include requesting resources supplied by some other unit or requesting an input from some other unit. Alternatively, the need might be only to synchronize the lifelines of the two units.

The distinguishing feature of coordinated interactions is their apparent reciprocity. If the request and reply interactions were to relate the subject unit to one external unit, the interactions would be truly reciprocal. However, it is possible for the subject unit to generate the stimulus interaction at one port and for the corresponding response to occur at another port. In this case, the subject unit could not distinguish whether the external unit that it stimulated is the same external unit whose response ultimately returns to the subject unit. The unit would still perceive the coordinated interactions as reciprocal.

NOTE 5—In information systems, this request-reply collaboration pattern is called *client-server*.

In contrast, a stimulus generated by the subject unit without anticipation of reply would simply be a part of the unit's response. Similarly, a response generated by an external without the initiating stimulus and anticipation by the subject unit would simply be another stimulus to the subject unit.

When a unit response includes stimulating a subordinate unit to generate a response interaction in return, the subject unit is imposing an *obligation* on its environment. As for available interactions, whoever or whatever uses the subject unit for delivering an intended effect is obligated to ensure that the necessary subordinate unit is available for replying to the service request from the subject unit. Failure to meet that obligation in a timely manner with an expected response prevents the subject unit from delivering its intended behavior.

NOTE 6—The identification of an obligation states an assumption in the behavior specification for the subject unit about a behavior of some other unit. That assumption is to be verified in the behavior specification of the other unit when that unit is identified in a build structure.

Figure 7 summarizes the concepts and terminology in this clause. The boxes labeled U_n are unidentified external units. The small black circle marks the head or tail of the interaction arrows to identify which unit controls the occurrence.

— The properties of input interactions to a subject unit are always controlled by their external source (U_1, U_3, and U_5); the properties of output interactions are always controlled by the subject unit (U_2, U_4, and U_5).

— The occurrence of a stimulus interaction (either input or output) is always controlled by an external unit (U_1 and U_2). The occurrence of a response interaction (either input or output) is always controlled by the subject unit (U_3, U_4, and U_5), but the reply from a subordinate unit (U_5) is expected.

— Interactions with peer units are uncoordinated (U_1, U_2, U_3, and U_4), interactions with subordinates are coordinated (U_5).

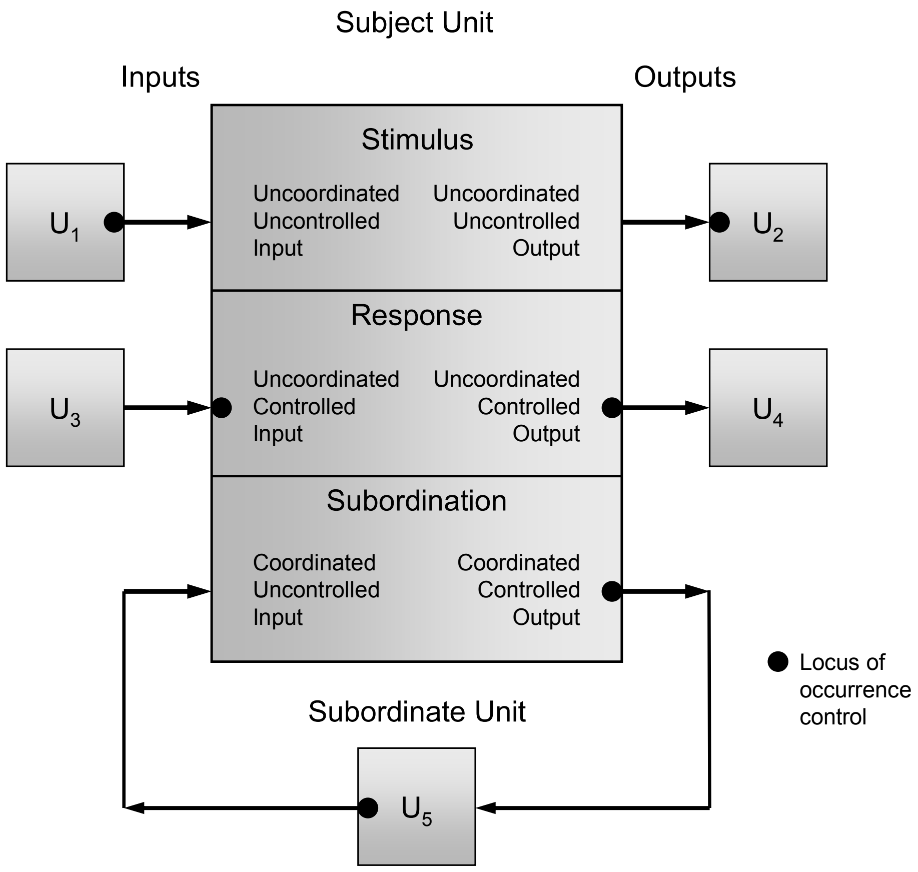

Subject Unit
Inputs
Outputs
Stimulus
Uncoordinated Uncontrolled Input
Uncoordinated Uncontrolled Output
U_1
U_2
Response
Uncoordinated Controlled Input
Uncoordinated Controlled Output
U_3
U_4
Subordination
Coordinated Uncontrolled Input
Coordinated Controlled Output
● Locus of occurrence control
Subordinate Unit
U_5

Figure 7—Inputs and outputs, stimulus and response, and control coordination

Thus, there are three cases of interaction occurrence control.

— *Stimulus*—An external unit controls an input interaction or an output interaction that the subject unit allows.

— *Response*—The subject unit controls input interactions and output interactions that external unit(s) allows.

— *Coordinated*—The subject unit controls a stimulus interaction to a subordinate external unit in anticipation (by design) that an external unit will generate a response interaction to the subject unit.

Consider the following examples:

— Waveforms within a certain frequency range appearing at an input port are amplified and appear at an output port. (Exemplifies stimulus input and response output.)

— An electrical power supply provides a regulated voltage from a battery unit. Current flow is available whenever a device on the power bus wants to take it. Ignoring its natural degeneration, the electromotive power of the battery decreases solely due to power withdrawal. Withdrawal is not under control of the battery unit, but it changes the state of the system. Another active current supply on the power bus (generator) can force current into the power supply to recharge it, increasing the stored energy. Restoration is also not under control of the battery unit. (Exemplifies stimulus output and stimulus input.)

— Driven by a balance of pressure, a hot water heater delivers an amount *A* of water of a certain temperature T_0 whenever the environment opens the output valve. Under an imbalance of pressure, the unit takes an equal amount of water *W* from an available external supply. As temperature drops, the unit also takes an amount of energetic material *M* from the environment and generates heat to warm the water to an operating temperature T_0 specified under control of the environment. The unit discards unused heat *U* into its environment. (Exemplifies stimulus output *A*, stimulus input T_0, multiple response inputs *M* and *W*, and response output *U*.)

— A simple inventory system accepts inputs for goods received *GR* and for goods sold *GS*. It accepts a request *REQ* for current inventory levels and produces a report *REP* containing that data. (Exemplifies multiple stimulus inputs *GR*, *GS*, and *REQ*, and a response output *REP*.)

— An embedded software system uses flash memory for retaining its data state when it is powered down. A technician flashes the memory, clearing it. This interaction by the technician is uncontrolled by the unit. In the context of this description, it is not useful to model this as an electromagnetic input phenomenon affecting electrical properties of the material; rather, it would be modeled as a stimulus that removes data from the unit. (Exemplifies a stimulus output.)

A stimulus interaction has either of two possible effects on unit behavior. The first (and most common) effect is to initiate a particular behavior occurrence of the unit. The second effect is to terminate a particular behavior occurrence. The notion of termination here does not necessarily include catastrophic or unexpected failure modes of the system. Instead, the focus is on explicitly required and implemented behavior termination functionality that is offered to the environment through interactions at the unit boundary.

NOTE 7—The external termination of actual behaviors for many real and complex units is often acceptably unpredictable. The reference model is concerned with specifying predictable outcomes when such are necessary. The use of a behavior model for investigating and understanding variant unit lifelines requires knowledge of the expected results of termination functionality when it is present. The purpose here is to ensure the construction of implied unit lifelines from the specification of unit behavior patterns.

Normally, a behavior occurrence runs its course and self-terminates in the unit. The notion of external behavior termination is that a particular behavior occurrence would end earlier and have a different response than would prevail in the absence of the external termination event. The behavior pattern for an externally terminated occurrence is thus a different pattern than for a self-terminated occurrence. These patterns differ in either or both of the following two respects:

— The duration of the two behavior occurrences might be specified differently.

— The causal and affective interaction relationships might be specified differently.

For a complex behavior whose occurrence can be externally terminated, there may be different outcomes depending on when the terminating event occurs. Several behavior patterns may need to be defined with different occurrence conditions on the timing of the termination event so as to define the variety of deterministic responses to be exhibited to the environment.

In real units, for behavior patterns involving only point-time interactions, the notion of termination is ambiguous. A behavior might have a duration that ranges from a point time to an arbitrarily long interval. While the duration of a behavior occurrence might be required to have some upper limit, it might not be required to have any lower limit. Without a minimum response time, it would be physically unlikely for a termination event to occur before the normal termination of the behavior occurrence. However, with a required lower limit to response time (i.e., a minimum duration of the behavior occurrence), then that duration would provide a time interval during which the termination event could sensibly occur.

The meaning of a termination event is likely to have more significance for behavior patterns involving extended time interactions. In such patterns, there is some ongoing unit behavior that continues for the duration of those interactions. Terminating the behavior necessarily means terminating those interactions. Specifying a termination event implies that the unit has either availability or occurrence control over the extended interaction. If an external unit has occurrence control over the extended interaction, then availability control at the unit’s port can be applied to terminate interaction, thereby terminating behavior at some time prior to occurrence termination. If the subject unit has occurrence control over the extended interaction, then it can terminate both the interaction and the behavior.

The result of termination is definite and specified in an appropriate termination behavior pattern (see 6.3.10). That specification may also include the occurrence of other response interactions, as well as appropriate changes in unit state.

6.3.3 Internal stimulus and response

Interactions controlled by external units are not the only stimuli that can initiate or terminate unit behavior. Similarly, interactions controlled by the subject unit are not the only responses that can be made by unit behavior.

An internal stimulus does not have any content associated with it, as that content would be, by definition, unobservable. An internal stimulus is an event within the unit that causes a result that is correlated with externally observable behavior. Internal event occurrences may initiate or terminate observable unit behavior. Not all internal events arising from the operation of a particular unit implementation will have meaning to the externally observable behaviors of the unit. But, for those that do have such meaning, the reference model provides concepts for specifying that meaning. The internal events of interest are those that are directly related to interaction occurrences at the unit boundary. While certain internal events involving timing might be associated with any behavior, there is a greater variety of events that can be defined through observables in terms of time-dependencies in extended time interactions and extended time behaviors.

The existence of timekeeping capabilities in a unit (see 4.3) allows the definition of event types whose occurrences are of periodic pattern, of prescribed pattern, or of single-shot time delays relative to interaction begin or end events. These events occur as the unit tracks passage of time, and their occurrences can be expected in the environment in accord with their defining rules in the unit. The mechanism for their generation is not visible, but the behavior occurrences they initiate or terminate are observable.

NOTE 1—A common use of timed internal events is for predictive coordination. When the unit delivers some service into the environment and there is some follow-on sequence of additional behaviors, there might be some delay specified between the end of one behavior and the beginning of the next to allow time for the environment to incorporate the effects resulting from the previous unit response.

Many engineered systems are designed with operational goals (fixed or adjustable) and some autonomous behaviors they can use for achieving those goals. Such systems do not necessarily need external stimulation to initiate or terminate some of their behaviors.

NOTE 2—As an example, consider a homeostatic unit that has been activated with a target value for some measured internal property that affects an output interaction. As various stimuli to the process component cause changes in the measured variable, the control component watches for out-of-limit deviations and activates some internal processes to raise or to lower the value of the measured variable toward the target value. The internal processes need different available inputs for their function. What is observed is the activation and deactivation of those processes without external stimulus.

During extended time behaviors, a unit might experience a large number of internal changes. Many of these changes are "accidents of implementation" to which observable behavior is indifferent. However, some changes might have a direct correlation with observable interactions (see 6.3.7). When there are particular times at which these time-dependent changes satisfy a particular externally meaningful criterion, these constitute events that can exhibit effects on behavior. Being meaningful in the environment, and sensible in their effect on observable behavior, these events are defined and their effects are specified in behavior patterns. For example, consider the accumulation of some amount of energy, material, or information through an extended time input interaction, without a concomitant removal by some output interaction. When the event occurs at which the internal property accumulation of the unit has reached a specified amount, this internal event might terminate the accumulation behavior, or it might initiate some mitigation response, whichever is deemed appropriate for that situation.

A common situation in which internal events are recognizable involves the chaining together of a succession of selected behaviors. If the situation has one initiation event in a certain context, and that leads to a lengthy sequence of output response interactions, only one behavior pattern is needed for its description. Besides identifying the required context and the initiation event, the behavior pattern enumerates the entire collection of response interactions and states the conditions that are met by the sequence and timings of the occurrence of the collection members.

If the situation has one initiation event in a certain context, and if that event leads to a lengthy sequence of input and output response interactions for which alternative behavior patterns might be necessary in order to account for necessary variability, then a more complex set of behavior patterns is needed. The overall behavior can be separated into alternative sequences of alternative behaviors, depending on the characteristics of the various interactions and changes in the unit that carry affective influences from earlier interactions to later interactions. The reference model allows specification of sets of alternative behavior segments. A selection and composition logic provides for constructing the various complex behavior patterns that are actually possible on the unit's lifeline from the various defined behavior segments.

What is unique in this situation is that only the first behavior segment is initiated by an external stimulus. The successive behavior segments (as appropriate to the context in which they would occur) actually do occur, by design or by control, as the unit continues operation. That is, the end event of a preceding behavior segment is an internal event whose effect is to initiate the appropriate follow-on behavior segment. The internal event is not directly observable, but its initiation effect is observable. At the end of each behavior segment, there could be multiple follow-on behaviors available depending on the circumstances at that point. The one that is appropriate for the actual circumstances is the one that actually occurs.

There can be behavior responses by the subject unit other than the generation of external interactions. Many engineered systems are designed with modifiable characteristics and behaviors that allow changes in the unit itself, within some limits.

NOTE 3—Most hand calculators have keys for Store to Memory and Recall from Memory. Pressing the Store key provokes an output indicating that something has been stored. However, the calculator response also includes a change to the calculator itself. The change does not affect most following operations of the calculator, but pressing the Recall key at a later time produces the stored display again. Alternatively, winding a clock changes the clock itself by storing the energy from the winding interaction. Later responses of the clock remove some of the stored energy, also changing the clock.

In some cases, an internal response to an external stimulus is changing the unit over time, and at some point the satisfaction of a particular internal condition might initiate occurrence of another behavior. Such situations need to be specified carefully and analyzed to ensure that conflicting behaviors are not specified.

In 6.3.7, the notions of behavior state and property state are introduced to account for the changes that can take place in a subject unit as a result of stimulated behavior. Such changes also affect the context in which responses are generated for stimuli.

Generally, the stimulus-response behavior of a subject unit exhibits an intermingling of both externally and internally stimulated forms.

6.3.4 Causal and affective relationships

Given that interaction occurrences are the percepts of unit behavior, the concepts of unit behavior are patterns that are intended, or are recognized, as governing the occurrences of unit response interactions based on unit stimulus interactions. A behavior pattern is a relationship that maps sequences of stimulus interactions to sequences of response interactions.

A behavior pattern description identifies correlated observations of the causal and affective relationships that characterize the interactions of a unit with its environment. The description of a behavior pattern is not a description of some processing mechanism by which that pattern can be generated.

A simplified statement of the causal aspect of behavior *B* is the following:

$$B\text{: } \mathit{Stimulus} \rightarrow \mathit{Response}$$

Read this as "Behavior *B* is a causal mapping (an implication) from the occurrence of a stimulus matching *Stimulus* to a response matching *Response*." The interpretation of this relationship statement is this: whenever an occurrence of *Stimulus* is present then an occurrence of *Response* is produced (eventually). Focusing on the order of interaction occurrence, this is described as *Stimulus causes Response*. This is a left-to-right reading of Figure 7.

The relationship between *Stimulus* and *Response* is a *causal relationship*. This is an existence dependency. This statement is a logical implication of sufficiency (not necessity) because it does not preclude the possibility that a different stimulus, *Stimulus′* for example, could also provoke the same *Response*. The stimulus-response pair defines the behavior occurrence *B*.

The complexity of behavior is in *Response*, as described in 6.3.2 and 6.3.3. In particular, a response could include one or more of the following events:

— Cause inputs of available material, energy, or information interactions to be taken from external units

— Generate stimuli to request service from subordinate units

— Receive responses from subordinate units

— Generate output interactions

— Change something internal to the unit (this is addressed in 6.3.7 and 6.3.8)

The causal relationship does not imply a functional dependency between the input and output interactions of the behavior. In the causal relationship, the content of output interactions might be completely independent of the content of any input interactions. The causal relationship is simply one of effective existence—under specified conditions, the existence of the stimulus causes the existence of the response. The causal relationship answers the question: How does some other unit elicit this particular behavior response from this unit?

There could be (and often is) a functional dependency associated with the causal relationship. This is a relationship on the content of those interactions. It corresponds to a top-to-bottom reading of Figure 7. In general, looking through *Stimulus* and *Response*, there is some collection J_I of *M* input interaction occurrences

$$J_I = (I_0, I_1, ..., I_{M\text{-}1})$$

and some collection J_O of *N* successor output interaction occurrences

$$J_O = (O_0, O_1, ..., O_{N\text{-}1})$$

where I_m and O_n are the input and output occurrences, respectively. Functional dependency means that J_I *affects* J_O. This influence relationship between J_I and J_O is an *affective relationship*. The (possibly implicit) functional relationship on behavior *B* is an $N \times M$ function.

$$\mathit{Function}_B(J_O, J_I) = 0$$

The properties and events of an output occurrence can depend on just one input occurrence at a single port or on several input occurrences scattered over multiple ports. Conversely, a single input occurrence might affect just one output occurrence or it might affect several output occurrences scattered over multiple ports. The affective relationship answers the question: How do the outputs of this unit depend on the inputs to this unit?

A substantial part of the complexity of engineered systems within the scope of this reference model is the historical dependency among their behaviors. This dependency is manifest in the observations that occurrences of identical stimuli at different times in the lifeline of a unit do not give the same responses. This can be expressed through an extension to the stimulus-response implication to incorporate an explicit dependence on the unit's history. Schematically, for the occurrence of the stimulus S_j at time t_j, the behavior B_j produces response R_j in the context of the history of previous behaviors on the unit's lifeline.

$$B_j\colon S_j \wedge \mathit{history}_{j\text{-}1}\,(B_0, B_1 \ldots B_{j\text{-}1}) \rightarrow R_j$$

Read this as "Behavior B_j is a causal mapping from the occurrence of a stimulus matching S_j at a point in the lifeline matching $\mathit{history}_{j\text{-}1}(\bullet)$ to a response matching R_j." This implication allows a different history to cause a different response to the same stimulus. The history function is not an actual record of all past behaviors on the lifeline. Rather, it is the set of conditions necessarily established by previous lifeline behaviors that allow the unit to respond as specified to this stimulus. The implication above is not an "executable" statement to determine what R_j happens when some S_j occurs in the context of a particular history. Rather, it is a statement of recognition that R_j is the result when S_j occurs in the context of particular conditions.

Of course, the occurrence of B_j changes the accumulated history of the unit, too.

$$B_j \wedge \mathit{history}_{j\text{-}1}\,(B_0, B_1 \ldots B_{j\text{-}1}) \rightarrow \mathit{history}_j\,(B_0, B_1 \ldots B_j)$$

These expressions show not only the causal dependency of R_j on the behavior history, but also the possibility of affective dependency of the outputs of B_j on the history of inputs. (This is elaborated in 6.3.7 and 6.3.8). Affective relationships might not only exist between the outputs and inputs of one behavior occurrence, but might also exist between outputs of one behavior occurrence and the inputs and outputs of previous behavior occurrences.

In summary, the description of behavior specifies both the causal and affective relationships between input interaction occurrences and output interaction occurrences.

— Causal relationships are variations on the following form: In certain situations, the occurrence of a particular kind of stimulus interaction is always followed by the occurrence of some particular kind(s) of response interaction.

— Affective relationships are variations on the following form: When properties of particular input interaction occurrences are changed, then properties of particular output interaction occurrences are changed.

6.3.5 Behavior pattern contents

Given that interaction occurrences are the percepts of unit behavior, the concept of unit behavior is a pattern that is intended as, or is recognized as, governing the occurrences of various input and output interactions. A behavior pattern is specified with an action statement. An action statement describes one distinct pattern of input interaction occurrences that causes the unit to produce a distinct pattern of output interaction occurrences. The collection of all such patterns constitutes a set of unit behaviors from which all possible evolutionary paths (lifelines) of the unit can be constructed in accord with particular sequencing constraints. These constraints are embedded in the specification of each unit behavior.

Actions are written in two parts. The first part is a pattern that applies to the stimulus, all input interactions, and salient features of the behavior history. This is called *Selection*. The second part is an assertion about the response, all output interactions, and amendment, as appropriate, of the behavior history. This is called the *Conclusion*.

The *Selection* is a unique, bounded partition of the circumstances in which the behavior described by the Action can occur. When a catalog of Actions is prepared for the unit, all possible circumstances of its use are to be found among the collection of *Selections*.

The *Conclusion* identifies the bounded range of results that are produced by the behavior, together with their dependencies on the elements in the *Selection*.

From the unit development perspective, the Action is a behavior capability to be provided by the unit. From the unit verification perspective, the Action is a chunk of behavior to be demonstrated. From the unit understanding perspective, an hypothesis about a set of circumstances can be found amongst the *Selections* in the catalog of behaviors; the corresponding *Conclusion* describes what is expected from the behavior occurrence.

The Action is a declarative statement of logical implication.

$$\textit{Action}: \textit{Selection} \rightarrow \textit{Conclusion}$$

The features in *Selection* include the following. (Compare with Figure 7.)

— Identification of the initiation event

— Identification of all input interactions and their causal controls

— Constraints on events and properties of the input interaction occurrences

— Constraints on the history of previous behavior occurrences

— Constraints between current input interaction occurrences and the history of previous behavior occurrences (dependencies on sequences and contents of these)

The features of *Conclusion* are stated in the context of a behavior occurrence satisfying the *Selection*. They include the following. (Compare with Figure 7.)

— Identification of all output interactions and their causal controls

— Constraints on events and properties of the output interaction occurrences relative to the *Selection*

— Relationships between the updated history of behavior occurrences and the history of previous behaviors

— Relationships between the updated history of behavior occurrences, the current output interaction occurrences, the history of previous behaviors, and the current input interaction occurrences

(The replacement of historical interaction sequences and contents with equivalent state representations is discussed in 6.3.7 and 6.3.8.)

The Action statement specifies an affective relationship among interactions past and present, with causal annotations to clarify the control and coordination requirements on the defined behavior.

The Action statement is not mechanistic; it has only a black-box interpretation. The interpretation is logical: if the situation described by *Selection* is met, then the situation described by *Conclusion* will prevail.

Thus, a behavior model does not contain an explicit identification of every possible occurrence of input sequence to output sequence mapping. Rather, it defines a collection of patterns that abstract all possible occurrences into distinctly differentiated Actions. It is clear from this introduction to behavior patterns that the Conceptual Metamodel contains a variety of forms for characterizing interaction events and properties, a history of interaction sequences, and relationships among these things.

The Action statement specifies one observable behavior pattern by which the unit transforms the current situation (known through input interactions), in the context of a given history, to a new situation (brought about through output interactions). The collection of action statements defines all controllability and observability properties of the unit.

The Action statement is declarative, not derivative, with respect to describing the behavior pattern conclusion when the behavior pattern selection is true. There is no description of mechanism for deriving the conclusion from the selection, only the statement that selection implies conclusion. In particular, the Action statement does not have an interpretation that all input interactions mentioned in the action selection occur before any output interactions mentioned in the action conclusion occur. In fact, the execution of a behavior pattern might actually exhibit any kind of interleaving of input and output interaction occurrences, subject only to the causal and affective constraints stated in the Action definition.

Just as an interaction occurrence has a begin event, so an action occurrence can be regarded as having a begin event. Specifically, the begin event for an action is the begin event for its initiating interaction. Unlike an interaction occurrence, there might be no obvious end event for an action. An end event could be identified as the point in time at which the behavior pattern is complete. This would not necessarily mean that all internal operations of some implementation of the unit have ceased, but it would mean that all elements in the behavior pattern occurrence have been accomplished. Clearly, the end event of an action cannot occur before all effects of the behavior pattern have been produced. A default definition for the action end event would be that moment at which the last conclusion element is established. Where particular sequences of inputs and outputs are required, constraints to that effect may be stated in the conclusion.

As mentioned in 6.3.9, the action end event might not need direct visibility to the environment if it serves to initiate follow-on actions whose occurrence is externally visible.

Usually, it is not necessary to designate an observation in the environment that corresponds with the end of an Action (see the end of 6.3.2). Most often, some “end event” is identified in order to record a performance specification. If the performance criterion is an upper limit between the initiation event and the appearance of a response interaction, for example, then the meaningful end event for the action is actually the begin event of the particular response interaction. A guarantee condition is written for that time difference between that event and the initiation event.

For behavior patterns having a termination event, the termination event marks the end of the action. This can be an external event or an internal event that terminates the action.

Under certain circumstances, an appropriate behavior pattern might be to provide no response at all to the stimulus. In effect, the end event of the Action is coincident with the begin event. (A stimulus that never has an observable effect on the unit’s behavior history or its environment is irrelevant in a behavior model.) The absence of any observable effect is hypothetically indistinguishable from an infinitely delayed end event. Certainly, an implementation of the unit would not be expected to implement this behavior as a delay.

As another alternative, the absence of identified output interactions could also mean the Action is only a change in the behavior history (a state change, see 6.3.4 and 6.3.5). A state change behavior (being the incorporation of interaction effects into its lifeline) is not directly observable itself, but there can be related events that are observable. For example, a change of some conservative state property might be correlated with the duration of the interaction causing the change of that property, so the interaction end event is a surrogate for the action end event. Alternatively, another behavior occurrence with input and output interaction occurrences might be initiated in the unit by the completion of the state change; the observation of that occurrence becomes an indicator for the occurrence of the preceding state change.

NOTE—The reference model in this standard does not provide for describing the moment-to-moment operation of the fabricated mechanism providing the realization of the unit behavior specification. Rather, it provides for specifying the "chunks" of behavior to be exhibited by the unit for various sequences and contents of stimuli and responses. Some particular realization might require concessions from the problem space (for example, frequency or size of inputs) but these are local to a few Actions. They do not affect the collection of behaviors required for supporting the lifelines of the unit in its application domain. They affect only the acceptability of a particular unit's implementation of these behaviors for meeting the application requirements.

For behavior to be elicited externally,

- Some externally caused interaction occurrence exists at the initiating event time.
- Internally caused interaction occurrences do not exist before the initiation event.
- Unless prohibited in the Action Selection, explicitly or through conflicting state change, behavior patterns may exhibit overlapping occurrence intervals.

While the action statement has no inference for more than this, any timing and content relationships between input and output interaction occurrences that are required of the unit can be stated in the selection and conclusion criteria. If a unit produces an output interaction occurrence to solicit some input from another unit, the designation of this output-input pair implies additional causal and affective constraints (see the discussion of obligations in 6.3.2 and 6.4.3). Explicit relative timing expectations may also be stated. In cases when there are observable "steps" that are to come first, second, etc., then separate action specifications are made for each.

The structure of component interactions in an aggregate interaction induces a similar structure among the actions selected by either the aggregate stimulus interaction or by the component interaction stimuli. If an aggregate interaction is defined with component interactions and used as a stimulus in a behavior specification, it is because an occurrence of the aggregate stimulus interaction selects one action statement to describe any aggregate conclusion, while occurrences of the components in the aggregate interaction select other actions to describe individual component conclusions.

- An aggregate interaction with sequential components induces a sequential selection of component actions following the selection of action by the aggregate interaction. There is no implication that component actions conclude in the same sequence as they are selected, unless explicit postconditions express that rule.
- An aggregate interaction with concurrent components induces all selected component actions concurrent with the selection of action by the aggregate interaction. Again, there is no implication that component actions conclude in the same sequence as they are selected, unless explicit postconditions express that rule.

When an action specification relates time-extended input and output interaction occurrences, it is regarded as a continuously occurring transformation. When an action specification relates point-time input and point-time output interaction occurrences, it is a point-time transformation. The Action specification is not restricted to just these two cases. For example, a time-extended input occurrence might be monitored for some particular input interaction property value, and the occurrence of this value could then initiate a point-time output interaction occurrence.

6.3.6 Typical behavior pattern forms

The Conceptual Metamodel provides general expressions for unit actions, for unit interaction history, and for observable constraints and enabling conditions. This clause identifies some stereotypical behavior patterns expressing causal and affective relationships that can be expressed in the metamodel.

Pattern 1: Contemporaneous stimulus-response pattern

In this pattern, an occurrence of input stimulus interaction *S* has a direct cause-effect relation with an occurrence of output response interaction *R*. There is a complete affective relation between the stimulus content *s* and the response content *r*. Assuming the stimulus occurs at t_0 and the response occurs at t_1, Action A1 could be written symbolically as follows:

$$A1: \quad S[t_0] \rightarrow \quad R[t_1] \wedge r[t_1] = function(\, s[t_0] \,)$$

Application example: This pattern could describe a simple functional response action.

Pattern 2: Content-dependent alternative contemporaneous stimulus-response patterns

This is a two-action pattern. In the first action, an occurrence of input stimulus interaction *S* has a direct cause-effect relation with output response interaction *R*1 (content *r*1). In the second action, an occurrence of the same input stimulus interaction has a direct cause-effect relation with output response *R*2 (content *r*2). The choice of resulting output response depends on the truth of some condition named *Condition0*(•) on the stimulus content. The two actions could be written symbolically as follows:

$$A1: \quad S[t_0] \wedge Condition0(\, s[t_0] \,) \rightarrow \quad R1[t_1] \wedge r1[t_1] = function1(\, s[t_0] \,)$$

$$A2: \quad S[t_0] \wedge !Condition0(\, s[t_0] \,) \rightarrow \quad R2[t_1] \wedge r2[t_1] = function2(\, s[t_0] \,)$$

where the symbol ! stands for negation and the symbol ∧ stands for logical AND.

Application example: This pattern could describe one functional response for inputs satisfying some constraint, with a different functional response for inputs not satisfying the constraint.

Pattern 3: History-dependent alternative contemporaneous stimulus-response patterns

This is also a two-action pattern. In the first action, an occurrence of input stimulus interaction *S* has a direct cause-effect relation with output response interaction *R*1. In the second action, an occurrence of the same input stimulus interaction has a direct cause-effect relation with output response *R*2. The choice of resulting output response depends on which particular condition, *Condition1*(•) or *Condition2*(•) is true on the history $history^B[t_0]$ of inputs prior to t_0. The two actions could be written symbolically as follows:

$$A1: \quad S[t_0] \wedge Condition1(\, history^B[t_0] \,) \rightarrow R1[t_1]$$

$$A2: \quad S[t_0] \wedge Condition2(\, history^B[t_0] \,) \rightarrow R2[t_1]$$

where the symbol ∧ stands for logical AND. In the Conceptual Metamodel, interaction occurrence histories are actually represented with a behavior state model.

Application example: This pattern could describe a pulsed toggle control; when its history of on-off signals contains an even number of pulses (*Condition*1), the output *R*1 is an "off" signal; when its history contains an odd number of pulses (*Condition*2), the output *R*2 is an "on" signal.

Pattern 4: Delayed effect stimulus-response pattern

In this action, an occurrence of input stimulus interaction *S* at port *P* has a direct cause-effect relation with output response interaction *R*. There is a complete affective relation between the response content *r* and the contents of some number of previous stimulus interactions. Assuming the stimulus occurs at t_0 and the response at t_1, action A1 could be written symbolically as follows:

$$\begin{aligned} A1: \quad S[t_0] \rightarrow \quad & R[t_1] \wedge \\ & \{\, r[t_1] = \mathit{function}(\, s[t_0], s_{-1}, s_{-2} \ldots), \mathit{or} \\ & \; r[t_1] = \mathit{function}(s[t_0], \mathit{history}^B[t_0]\,)\,\} \end{aligned}$$

where $s[t_{-n}]$ denotes interaction content of previous occurrences of S. (In the Conceptual Metamodel, interaction content histories are actually represented with a property state model. See 6.3.7.)

Application example: This pattern could describe a simple database query in which the output content is the value received and stored previously for the same key.

6.3.7 States and interaction history

A significant feature in many engineered systems, particularly computer-based systems, is the ability to deliver various behaviors into the application domain at one point in time, based on the history of behaviors elicited and delivered at previous points in time. Such delayed effects in system responses are among the "affective behaviors" described in 6.3.4. Such effects are represented most commonly with some form of "state" metamodel. The reference model includes a notion of "states" for describing the dependence of behaviors on the history of previous unit behaviors.

Ascribing state to a unit implies that it has a capability to modify future behaviors as a result of past interactions with its environment. In particular, it has the capability to modify itself as the result of one behavior, and it can use this modification at a later time to affect elicited behaviors, either in terms of the behaviors that are available for elicitation or in terms of the output interactions produced by those behaviors. This capability is frequently implemented as a passive storage element whose contents can by modified ("the state") and active elements that modify and sense the content in that storage element ("the state transitions").

There are a great variety of state model representations, each useful for particular purposes. Two categories of state models are distinguished in the kinds of states being modeled: *white-box* state models that represent states of an interacting collection of objects, and *black-box* state models that represent states of a single object.

— White-box state models describe the coordinated operations of a collection of objects (or system design elements). States are composite descriptions over the collection. Changes in state result from interactions of some object behaviors in some configuration. Sometimes the states are application-focused (e.g., the objects are commuter trains traversing a network of tracks and stations with arrival and departure synchronization). Sometimes the states are mechanism-focused (e.g., a collection and their interacting behaviors). The states are often states of the configuration and not directly traceable to states of the individual elements.

— Black-box state models describe the history-dependent operations of an individual object. The states are not necessarily related to any internal design elements or structure of the object; they are defined precisely by the observable effects they have on object behavior.

The black-box state model is appropriate for use in the reference metamodel for behavior specification.

There are two principal patterns of historical behavior dependency exemplified by Pattern 3 and Pattern 4 in 6.3.6.

— Pattern 3 illustrates how variations in the history of input interaction occurrences result in the unit providing entirely different responses for the same stimulus. This pattern is described with a "behavior state metamodel." A common example of a behavior state metamodel is a "finite state machine."

— Pattern 4 illustrates how variations in the history of input interactions do not affect the pairing of a particular stimulus interaction with a particular response, but they do affect the content of the response occurrence. This pattern is described with a "property state metamodel." A common example of a property state metamodel is a "retained data model."

In a behavior state metamodel, the states represent a partitioning of all system behavior patterns into (possibly overlapping) subsets of behavior patterns that can be elicited from a unit at a particular point in time. One subset at a time is "active," meaning the unit notices and responds to stimuli that initiate behaviors in that subset only. Following the occurrence of certain special stimuli, a different subset of behavior patterns becomes "active" and the previous subset becomes "inactive." This change of active subsets of behavior patterns is called a *state transition.*[8] In the given interpretation of Pattern 3 (see 6.3.6), Actions *A1* and *A2* are assumed to be in two different subsets of behavior patterns, for example *S1* and *S2*. Then *Condition1*($history^B[t_0]$) asks these questions: Has there occurred any pattern of previous behaviors that makes *S1* active, and has there not occurred since that time any pattern of previous behaviors that make *S1* inactive? A similar question is implied for *Condition2*($history^B[t_0]$).

In a property state metamodel, one or more properties (either conservative or non-conservative) are defined and associated with the unit itself. These are state properties and they are not directly observable as their implementation would lie within the boundary of the unit. Rather, they provide an accounting basis for relating behavior interaction occurrences at earlier times to behavior interaction occurrences at later times. These state properties are functions of the history of previous interaction occurrence properties, and in turn, values of current interaction occurrence properties are functions of the values of the current state properties. Property state accounting requires the definition of the two functional relationships that provide the delayed affective connection between prior inputs and later outputs. (An important consideration for first use of a unit is the specification of initial values of the state properties that would be necessary for initial unit behavior.) A particular property state is determined at a point in time by the collection of current values for all state properties.

For a conservative property such as mass, behavior patterns might specify a contemporaneous balance of input and output interaction amounts exchanged between the system and its environment. However, if contemporaneous balance is not kept between inputs and outputs, then a delayed affective behavior could describe property accumulation or depletion by using a conservative property state variable.

NOTE 1—A state variable TotalMass could be defined and could be changed as a result of input and output interaction occurrences when various behaviors occur. For a particular behavior occurrence, a mismatch of property amounts (or rates) between input interaction occurrences and output interaction occurrences would result in increases or decreases in the value of the variable.

Several characteristics need consideration in a conservative property state model, including possible durations of mismatched property exchange, maximum aggregation amounts, and additional behavior patterns for excess property input and property state depletion.

NOTE 2—For a series of batch transfers, the functional relationship giving value to the variable TotalMass is just a summation of all mass transfer amounts. If interactions were characterized by flow rates and time durations, then the functional relationship would be a time-based integration of flow rates.

Delayed affective behaviors involving information interactions between a unit and its environment also need a property state model for their description. While information need not be a physically conservative property, its property state model can exhibit a complex combination of non-conservative and conservative elements.

The simple information system is an archetype for a unit with a non-conservative property model. For some application domain of concern, information is to be collected at some points in time and used to answer queries posed at different (later) points in time. The property state model is defined by an appropriate retained data model (passive, non-conservative, holding most recent value) and associated create, retrieve, update, delete behaviors (active, conservative tables of records).

[8] This relationship between actions and states is characterized as a Mealy state machine.

NOTE 3—A personal desktop calendar system provides several examples of information property states. Appointment records accumulate on entry and deplete on deletion (conservative). Irrespective of the number of times a date and time and reminder interval are input for an appointment, only the most recently entered values are retained (non-conservative). Retained appointment records move conservatively from pending to past. Record status between imminent and occurring appointment times is repeatedly announced (non-conservative).

Aggregate property states may also be defined. Conditions that specify particular constraints or relations can be used to distinguish subsets of the collection of property state values. The condition is an invariant of the aggregate state because all members of the subset satisfy the condition. (The computational data state and the invariants of structured programs are of this form.)

From a behavior state perspective, the occurrence of a certain interaction at one time changes, for later times, the association of output interaction instances to input interaction instances. From a property state perspective, the occurrence of certain interaction properties at one time changes, for later times, the occurrence of other interaction properties, but does not alter the existing input-output interaction association pattern.

6.3.8 State observation

The reference metamodel provides a shorthand representation for a history of behavior occurrences. This shorthand representation is a state metamodel. The explicit goal of this state representation is to specify directly the delayed causal and affective behavior of the unit, avoiding any description of possible implementation mechanism features or operations. State is thus an accounting notion that spans time on the unit’s lifeline.

State is a temporally localized summary (the state “now”) of the effect of previous behavior on the unit. *Temporally localized* means particular elements of state are changed during a behavior occurrence in response to particular interaction occurrences, and those same elements are examined or used at a later time.

There are two stereotypical forms of delayed affective behavior, as follows:

— Changes in behavior patterns accessible to the unit’s environment at different times

— Effects of earlier behavior interaction content on later behavior interaction content

These behavior effects can be described with some common state metamodels. Behavior state models are most often represented with some form of Finite State Model, possibly hierarchical. Property state models are most often represented with some form of Retained Data Model (e.g., a relational data base logical model). These are not the only representations that can be used. In fact, a behavior state model can be represented in a structure of retained data, and a property state model can be discretized into a finite state model.

Abstractly, the states before and after Action A_0 are as follows:

$$State[\ t_0\ ] = function_0\ (\ A_{-1}, A_{-2}\ ...\ )$$

$$State[\ t_1\ ] = function_1\ (\ A_0, A_{-1}, A_{-2}\ ...\ )$$

where A_{-n} indicates the n^{th} preceding action. This interpretation of state does not necessarily mean that the unit is actually changed with every Action occurrence, only that it could be changed. In actual units, for example, simple functional behaviors have no effect on the unit; the state function simply ignores those interaction occurrences.

When $State[t_0]$ and $State[t_1]$ are different, they are labeled with the modified terms *current state* and *next state* and given the following black-box interpretation:

— *Current state* is the history of significant behavior occurrence sequences antecedent to the end of a particular behavior occurrence, which is to say the current state cannot exist after the end of the behavior occurrence.

— *Next state* is the history of significant behavior occurrence sequences subsequent to the beginning of a behavior occurrence, which is to say the next state cannot exist before the beginning of the behavior occurrence.

While the current state and next state can be distinguished, the reference model does not allow the temporal localization of any of the differences between them; that is, often there is no "event" marking the time at which a state transition occurs. Antecedent and subsequent are defined, but there is neither mechanism nor time point for the change from antecedent into subsequent.[9] States and state transitions are not directly observable outside the system, except as they influence subsequent behavior through allowed input interactions and generated output interactions. The Conceptual Metamodel is restricted to describing only this subsequent behavior. Design development produces assemblies of units having coordinated sequences of activities, and those details emerge in lower level specifications as the observables of lower level units. The behavior specification is the criterion for judging its design and not a vehicle for its expression.

Various shorthand representations for state are readily related to interaction occurrence history. For example, consider interactions with a physical mass property (normally conservative). There is a real and often reportable state property of accumulated mass. The inflow and outflow interaction history is directly accountable by summation or integration into the amount of accumulated mass at any point in time, so the accumulated mass is a useful replacement for the history of previous mass transfer interactions.

Similarly, one stimulus interaction might enable a subset of behaviors to be elicited by the environment, and a second stimulus interaction might prevent that elicitation. There might be no observable unit response to indicate either of those state transitions. However, attempting to elicit one of the behaviors in the subset will either succeed or fail, depending on which of the two stimuli have occurred most recently in the unit's lifeline. Behavior states directly represent this situation.

It is also possible for a defined state property to be used like a behavior state through the use of conditions. A condition applied to a state property is similar to a condition applied to an input interaction property. It partitions occurrences of the state with different property values into those for which the condition is TRUE and those for which it is FALSE. If behaviors are allowed when the condition is TRUE that are not allowed when the condition is FALSE, or vice versa, this condition would be included among the selection constraints in the Action definition. Alternately, two behavior states could be defined with that property condition, one state including those property values for which the condition is TRUE and the other including those property values for which the condition is FALSE. In such use, the condition is called a *state invariant.* (See the further discussion on conditions in 6.3.9.)

State definition can also interpose a functional relationship between interaction history and state properties. For example, it is possible that only a count of certain input interaction occurrences is relevant to future behavior of a unit.

What is essential is to ensure that features of a state model (behavior state, property state) are directly expressible as delayed affective observations between input and output interaction sequences. The reference model provides for describing behavior states and property states in a hierarchical form of Extended Finite State Machine metamodel.

First, a hierarchy of behavior state nodes is constructed to identify those subsets of unit behaviors that can be inhibited and offered at different times along the unit's lifeline. This is a hierarchy of finite state machines. They segregate subsets of behavior patterns accessible to the unit's environment at different times.

[9] If the moment of change seems to be needed in a specification, then perhaps behavior patterns are being specified at an unsuitable scale or perhaps mechanistic productions of behavior are being described rather than the end effect of the behavior.

Second, a property state of variables and functions is developed to account for how these are affected by behavior interaction contents and how these affect later behavior interaction contents. Property variables are either conservative or non-conservative with respect to input and output. Property variables could be simple kinds of data or they could be complex aggregations, structured through various associations, augmented with constraining conditions, and collected into set or bag structures.

Third, different properties or property collections are associated with different behavior states. A property state model associated with a behavior state can be referenced and modified by any actions in that behavior state or in its subordinate behavior states. The property state model is not "visible" to actions in peer behavior states or to actions in any aggregating super-state. In this way, the specification can indicate the extent of affective influence by the property state. Most commonly, this extent of influence is related to the duration of effect on behaviors or the scope of affected behaviors. Properties occur higher in a state hierarchy if they are longer lived or have a larger scope of effect.

6.3.9 Behavior pattern observation

The use of conditions to express and identify behavior discontinuities with respect to interaction properties and time is described in 6.2.5. Conditions in behavior patterns serve to relate and partition the composite domains of interactions and interaction histories (states) into subdomains in which distinct behavior patterns are to be exhibited.

These conditions fall into two groups. The first group includes *guard conditions,* those conditions that determine whether a particular behavior pattern occurs (the action selection in 6.3.5). The second group includes *guarantee conditions,* those conditions that constrain the results of behavior pattern occurrence (the action conclusion in 6.3.5).

Guard and guarantee conditions do not have specific temporal relationships with respect to the occurrence of a behavior pattern. (See 6.3.8 for a related discussion of the timing of state change during the occurrence of a behavior pattern.)

— A guard condition discriminates on the current input interaction occurrence(s) and current state. The observations of input interaction occurrence are regarded as being made whenever the inputs mentioned in the pattern exist, regardless of possible interleaving with output interaction occurrences.

— Satisfying the guard condition implies only that the behavior pattern conclusion occurs.

— The guarantee condition for this occurrence has the current input interaction(s) and current state as given elements; it relates the output interaction occurrence(s) to the current input interaction occurrence(s) and the current state. The observations of output interaction occurrence are regarded as being made whenever the outputs mentioned in the pattern exist, regardless of possible interleaving with input interaction occurrences.

— The guarantee conditions also relates a next state (if different) to the current input interaction occurrence(s) and the current state.

The expression of condition statements depends on the form of state definition used (i.e., state defined as history of previous input interaction occurrences or as a current account of previous interactions).

A guard condition is a logical conjunction of conditions of the following kinds:

— Constraints on events and properties of the input interaction occurrences

— Constraints on the interaction history in the current state (behavior and property)

— Relationships between current input interaction occurrences and the current state

A behavior pattern becomes a behavior occurrence when a behavior stimulus occurs, the unit is in an active state that allows the behavior, and the guard condition for the behavior is satisfied.

Whether or not a particular behavior is active at some point in time might be determined through observations of the unit along its lifeline. (It is not generally possible to determine whether a behavior is active through a single point-time observation of a unit.) The determination that a behavior is active is represented by a unary condition associated with a behavior pattern. That unary condition is TRUE for the duration of any occurrence of the behavior pattern. The time at which it becomes TRUE corresponds with the begin event of the occurrence. The time at which it becomes FALSE corresponds with the end event of the occurrence. Because the initiation of a behavior occurrence might need to account for other behavior patterns that could be active at the time of stimulus, the following kind of guard condition is added to the preceding list of guard conditions:

— Occurrence status of a behavior pattern, *occurring*(⟨*action*⟩)

The role of this guard condition in the general description of overlapping behaviors is addressed in 6.3.10.

A guarantee condition is a logical conjunction of conditions of the following kinds:

— Constraints on events and properties of the output interaction occurrences

— Relationships constraining the current output interaction occurrences in terms of the current input interaction occurrences and the current state

— Relationships determining the next state (behavior and property) in terms of the current input interaction occurrences and the current state

Internal changes in state, their sequencing, and their timing through the occurrence of a behavior pattern are not regarded as observable. If there are occurrences of specific input or output interactions that synchronize such property state changes with the environment or make them visible to the environment, then distinct action statements would specify explicitly the various behavior patterns and their sequencing.

Generally, conditions define the following kinds of observables:

— Restrictions on individual interaction occurrence properties

— Restrictions on individual interaction occurrence events

— Restrictions on individual state properties

— Relationships between interaction occurrence properties

— Relationships between state properties

— Relationships between interaction occurrence properties and state properties

— Relationships between interaction occurrence events

Furthermore:

— Restrictions may be existential or domain restrictive.

— Relationships may be comparative, associative, or quantified over collections.

Finally, conditions on time-varying properties can also define events. Identifiable points of time on which unit and environment behaviors synchronize include the following:

— Time of equality in comparison of individual time-varying interaction occurrence property values with fixed values (absolute or specified in state)

- Time of equality in comparison of individual time-varying state property values with fixed values (absolute or specified in state)
- Time of equality in comparison between concurrently existing time-varying property values in interaction occurrences and state

The occurrence of any of these events might initiate another behavior occurrence in the unit and might also terminate the current behavior occurrence. These are *internal stimuli*. They are not implied by the occurrence of external interaction begin or end events, though they cannot occur outside the context of an extended time behavior.

Intervals can be defined by inequality forms. Condition ($p1(t) \geq p2(t)$) defines an interval with a begin event when it becomes TRUE and an end event when it becomes FALSE. Such conditions can be useful for specifying intervals of continuous action occurrence.

The predicate for time of equality may include directional qualification, e.g., time of value crossing from LESS THAN to GREATER THAN.

An instance of the reference metamodel for specifying behavior has a vocabulary of defined model-specific terms for the following observables:

- Interactions with properties, begin events, and end events
- Input interactions and output interactions
- Histories of interaction occurrence sequences and properties expressed in a two-tier structure of behavior states and property states
- Conditions describing observable characteristics of individuals, pairs, and collections of these

The reference model emphasizes the use of guard and guarantee conditions in Action definitions. Careful analysis of these is crucial for capturing all the constraints to be satisfied by an implementation of a unit. Omissions in the behavior model lead to misdesign, mistesting, and misuse of the unit.

The combination of action, state, and condition elements in the reference model defines a form of predicated extended finite state machine.

6.3.10 Concurrent behaviors

The simple rule of specification interpretation is that an available behavior, when elicited by stimulus, will occur. That is, if the active state of the unit allows the behavior when the behavior stimulus occurs, and if the guard condition for the behavior is satisfied, then an occurrence of the behavior begins. If no other behaviors are initiated in the unit, the behavior occurrence continues until it accomplishes the inputs, outputs, and state changes specified in the applicable behavior pattern. Point-time behaviors have a single behavior pattern. Extended time behaviors may have two or more behavior patterns: one describes this "normal" occurrence and the others describe "terminated" occurrences.

If behavior-initiating events occur along the unit's lifeline at a rate slow enough that every behavior occurrence completes its response before the next initiation event occurs, then there will never be more than one behavior occurrence at any time along the lifeline. If this situation does not prevail for the unit, then some behavior occurrences are initiated before others end.

If there is no conflict of assumptions about the unit or outcomes between two behaviors, it is assumed that the concurrent or overlapping occurrences of each behavior are no different than they would be in the absence of the other behavior occurrence.

NOTE—This assumption always needs to be verified for any proposed design of the unit.

If there is a conflict of assumptions about the unit or outcomes from the behavior that might occur, the result is not a simple superposition of the individual behavior occurrences. Either or both of the individual occurrences is altered from what it would have been in the absence of the other.

Conflicts can occur in a variety of ways. Examples of conflicts between two concurrent behavior occurrences (called simply *the first* and *the second*) that start at the same time include the following:

- The response of the first is to pull an input interaction at a particular port; the response of the second is also to pull an input interaction from that same port.
- The response of the first is to accumulate an externally controlled input and increase a conservative state property; the response of the second is to discharge an internally controlled output and decrease the same conservative state property.
- The response of the first is to change the behavior state to a state in which the second can not be initiated.
- The first and the second have no conflict other than a resource limitation that lengthens their occurrence time; if this is unacceptable, the begin event of one or the other may be a termination event for the other (a priority override).

The composition of outcomes from each of these possibilities cannot be determined from the defined responses of the individual behaviors. When occurrences of conflicting behaviors have the possibility to overlap in time, there are several issues to be addressed and resolved before the intended behaviors can be specified.

- What behavior occurrence overlaps are possible? Which behaviors can have overlapping occurrences? How many behavior occurrences can overlap at any point in time?
- What dependencies between behavior occurrences should be maintained? What conflicts between behavior occurrences might exist?
- What characteristics of overlapping behavior occurrences should be constrained by specification?

When such conflict possibilities are probable, and an undetermined outcome is undesirable, care is needed in developing the behavior specification. There is no need for the conceptual model to provide a mechanism for behavior composition as that is a unit design problem. The specification concern is to describe the expected resulting behaviors and to indicate the circumstances in which each can be exhibited by the unit. The description of intended composition of overlapping behavior needs to be included in the collection of behavior patterns for the unit. This is accomplished as follows.

First, the individual behavior patterns are specified with guards that include an explicit disallowance of a concurrent occurrence of the other. This describes the intended behaviors in the absence of conflict. For potentially conflicting actions *A1* and *A2*, the amended description with the added guard conditions would be as follows:

```
A1a:    if ! occurring( A2 ) ∧ ( remainder of A1 guard );
        initiate by E1;
        { description of A1 response }

A2a:    if ! occurring( A1 ) ∧ ( remainder of A2 guard );
        initiate by E2;
        { description of A2 response }
```

where $\wedge$ is logical AND.

If there is an occurrence of *A*1 and the initiation event for *A*2 does not occur, the occurrence of *A*1 concludes as expected. A similar result is obtained for *A*2 occurrence when there is no initiation event for *A*1.

There are three possible situations for overlapping behavior occurrences of *A*1 and *A*2, as follows:

— *A*2 is initiated during an occurrence of *A*1

— *A*1 is initiated during an occurrence of *A*2

— *A*1 and *A*2 are initiated simultaneously (within the time granularity of the specification)

The descriptions of the first and second situation are identical under the exchange of labels so the first is addressed. There are two steps to describe what is to happen if *E*2 does occur during the occurrence of *A*1. The first step is to specify the desired compound behavior during the overlap of *A*1 and *A*2 occurrences. This composition of behavior, for example labeled *A*3, describes what is observed when each of the individual behaviors affects the other, with *A*1 having started first.

```
A3a:    if occurring( A1 ) ∧ ( remainder of A3a guard );
        initiate by E2;
        { description of required  A1 + A2 response }
```

At the time of *E*2, the first behavior occurrence on pattern *A*1 no longer describes the unit response. This requires an alternative to *A*1 with termination, as follows. (See 6.3.2 for the description of terminated behavior.)

```
A1b:    if ! occurring( A2 ) ∧ ( remainder of A1 guard );
        initiate by E1;
        terminate by E2;
        { description of A1 response interrupted by A2}
```

Along the unit lifeline, in the absence of *E*2, what would be observed is an initiated occurrence of behavior *A*1 with its defined response. If *E*2 occurs before the end of *A*1, what would be observed is an initiated occurrence of behavior *A*1a followed at the time of *E*2 by the occurrence of *A*3.

If the second case, initiation of *A*1 during an occurrence of *A*2, a similar addition of two more patterns (*A*3b and *A*2b) would be required to specify that outcome.

If *A*1 and *A*2 are indeed conceived as separate behaviors, the likelihood for simultaneous initiation can be lowered as much as desired by choosing a smaller temporal granularity. (Conversely, for a large enough temporal granularities, many behavior stimuli would be perceived as simultaneous.) Rather than attempt to address absolute simultaneity here, the foregoing approach illustrated with *A*1a, *A*2a, *A*3a, *A*1b, *A*2b, and *A*3b is used with a small enough temporal granularity for its applicability.

The variety of complex behaviors exhibited by concurrent behaviors of real systems is not the focus of description in the behavior model. Those behavior outcomes are often neither well-defined nor constrained, and in such cases the behavior of the unit could display confounding and conflicting results. Those results might not be what would have been needed for the successful delivery of unit capability into its context. They are, rather, design-induced results that were not subject to any specified behavior constraints. The behavior model focuses only on the observable description of desired behavior in these situations, using guard conditions and initiation and termination events to specify what is to happen on the unit lifeline.

6.4 Observables—Base quantities

The connection between the concepts of this reference model and the reality of a particular system's behavior is made through *base quantities.* These elements are used in the reference model to quantify a behavior model into a behavior specification. The base quantities provide the ground truth observables for empirical behavior quantification.

There are three kinds of base quantities, as follows:

— Property

— Event

— Condition

These quantities have been discussed previously. Their salient features are summarized here.

6.4.1 Observable property

An observable property is a measurable phenomenological characteristic of an interaction. The definition of a type of interaction specifies all such properties of the interaction that are meaningful to the behavior model. Normally, properties are regarded as characteristic traits or peculiarities of the interaction occurrences being defined.

A property definition has two aspects. Intensively, a property is associated with certain phenomena by which it can be detected and measured. A property has some scale of value and a unit of measure. When the property is intended to characterize some type of interaction, it is commonly given some domain of possible values. Property values need not be explicit and continuous; they may be discrete and they may be implied by pattern definitions. When a property is specified for some type of interaction, the interpretation is that all occurrences of that type of interaction do have that property. Each interaction occurrence exhibits the property, and the measurable value of the property for that occurrence is within the domain of possible values defined for the type of interaction that describes that occurrence.

Properties may be intensive or extensive. They may include such physical or logical attributes as the following:

— Volume

— Mass

— Pressure

— Temperature

— Composition

— Frequency

— Amplitude

— Amount or rate

— Encoding of information carried by patterns in physical property

— Encoded data length, type, and value

Properties may be grouped into application-meaningful collections and given a group property name.

Property values for a point-time interaction occurrence are constant. For extended time interactions, a property's value may be constant or time-varying.

There is no conceptual limit to the list of properties that may be used to characterize a type of interaction. Interdependent properties may have value ranges that are functionally dependent.

The properties used to define a particular type interaction are not just physical or logical properties of a particular interaction occurrence. They may also include the following:

— The characteristic time duration of the interaction occurrences, with some range of values

— Functional pattern(s) describing time-varying properties, with various descriptive features such as maximum and minimum rates of change, frequency of change, maximum and minimum values

While directly observable properties characterize interactions at the boundary of the unit, the Conceptual Metamodel also includes the notion of properties that characterize property states of the unit (see 6.3.7). These may be regarded as indirectly observable properties, as they account for the affective influence of interaction properties occurring in an interaction at one place and time on interaction properties occurring in an interaction at another place and/or a later time. While there is no mandate that the accounting of this influence have a particular physical basis, there is usually a reasonable choice of state properties having a straightforward relationship between the observable interaction properties.

State properties are defined in analogy with interaction properties. However, such definitions do need to account for any conservative or non-conservative characteristics of the affective relationship. For example, for a mass accumulation property in a unit's property state there could be an associated recording of the largest imbalance between input and output mass along the unit lifeline. That maximum mass of accumulation value might be much larger than the mass transferred by any individual input or output interaction occurrences.

6.4.2 Observable event

An observable event is a singular moment in time at which some perceptible phenomenological change (energy, matter, or information) occurs at the port of a unit. The change that defines an event is, by definition, smaller than the granularity of time assumed in a behavior model. An event is a time marker along the actual lifeline of a subject unit. An external event is a perceptible phenomenological change that can simultaneously mark time along the lifeline of both the subject unit and an external unit. It might be determined differently by the external unit and by subject unit, but the distinguished moment is common to the lifeline of both units.

As for properties, a behavior model only defines those events that are meaningful to the description of unit behavior. Events are not defined for every time-dependent change that might be used to mark the progress of time, but only when such times are associated with observable behavior occurrences or behavior changes.

An event definition has two aspects. Intensively, an event is associated with certain phenomena whose change marks its time of occurrence. The simplest kinds of change definition are when an interaction occurrence changes from absent to present or from present to absent. (There is at least one property of the interaction that is observable and exhibits valid values for the interaction during the time it is present.)

Defining the begin event or end event for a type of interaction occurrence has an associated extensive definition; namely, the set of begin (or end) times for all occurrences of that interaction at a particular place (a port.)

For an extended time interaction occurrence, there are begin and end events that would have distinct times for each interaction occurrence. If all properties of the interaction occurrence are constant, these are the only two distinguishable times associated with the interaction. However, when properties of an interaction occurrence are time-varying, there is another way to distinguish particular times along the lifeline between the begin event and the end event. Specifically, a sequence of observations (measurements) of the time-varying interaction property may be made, and the values may be compared with one or more threshold values. An event time may be defined for crossing of a threshold value in one or both directions. Again, the notion is that such events may be associated with various behavioral effects, perhaps terminating some behaviors and initiating others. (Events that have no meaning in describing unit behavior would have no need of definition.)

Observable interaction-based events provide reference points in time that are common to the subject unit and the external interacting units; other events may be defined relative to them through unit timekeeping capabilities, as described in 6.3.3. These include event types whose occurrences are of periodic pattern, of prescribed pattern, or of single-shot time delays

Further, as externally observable events may be defined on time-varying interaction properties, so internal events may be defined on time-varying state properties and used as described in 6.3.3 and 6.3.9.

For describing some related effects of concurrent non-interacting concurrent behavior, the reference model provides the following event structures. Each of these defines one event occurrence in terms of an enumerated set of different event type occurrences.

— The *firstOf*(•) structure defines an event whose occurrence is defined as the first event occurrence in the set. The event marks the earliest time at which any enumerated event occurs. This provides an "OR" notion for multiple event occurrences. This structure provides an event occurrence (point in time) that is marked by any of several alternative possible events.

— The *lastOf*(•) structure defines an event whose occurrence is defined as the last event occurrence in the set. The event marks the time at which all enumerated events have occurred. This provides a weak (accumulating) "AND" notion for multiple event occurrences. If those events are interaction end events or action end events, the event indicates when all listed interactions have concluded or when all actions have concluded. Such an event may be used to initiate follow-on behavior only after all items in the group of interactions or actions are complete.

— The *concurrenceOf*(•) structure defines an event whose occurrence is defined as the simultaneous occurrence (in the same time grain) of all enumerated events in the set. The event marks the common time at which all enumerated events occur. This provides a strong "AND" notion for simultaneous event occurrences. This structure provides an event marking simultaneity of occurrence.

6.4.3 Observable condition

An observable condition is a proposition or assertion about observables of behavior. A condition may be a statement about direct external observables or about indirect observables. Direct observables are interaction properties, interaction events, and action events. Indirect observables are properties and events whose existence, measurable values, and times of occurrence can be inferred through other observables; these include behavior and property state effects of unit behavior history and some internal property changes and events of extended time behaviors.

A condition is a statement that affirms or denies something about one or more individuals. There are two parts to the definition of a condition statement: the predicate and the subject. The predicate is a template that describes a property of objects or a relationship among objects represented by the subject. It identifies an observation or evaluation that can be made, such as "__ less than __." (The underscores "__" are placeholders for the subject elements.)

The subject identifies the individual(s) or surrogates to which the predicate observation is applied for determining the truth of the condition. In a behavior model, the individuals are occurrences of properties or events and relationships between them.

NOTE—In a statement like "*A* less than *B*," the predicate is "less than" and the individuals A and B might be two properties of an interaction, for example. The statement "*A* less than *B*" is not meant as a comparison of the definitions of the two properties, but rather a comparison of the actual values of the properties during a particular occurrence of the interaction. Thus, conditions are written for instances of a type, but they are evaluated at a point in time for particular occurrences.

Conditions on property values and event times are usually quantitative comparisons such as "__ less than __" and "__ equal __." Placeholders may be property instance names, event instance names, property functions giving values over times, or attributes of such functions.

Property and event conditions can be compounded through $\wedge$ (AND), $\vee$ (OR), and ! (NOT).

In addition to basic proposition forms, behavior specifications often require the use of first order logic to interpret predicates as sets defined on attribute commonality or as relations between sets, with existence and quantification over sets. This includes predicates such as *occurring*(•) mentioned in 6.3.9 and existential conditions on events that might have occurred on the unit history lifeline mentioned in 6.3.7.

In a behavior model, a condition serves to describe constraints that partition sets of possibilities into subsets, or subdomains. This is the usual case for guard conditions. A condition can also be regarded as affirming or denying a predicate of a subject. This is the usual case for guarantee conditions.

Although conditions appear as guards and guarantees in action definitions, they can be used effectively in definitions of interaction properties and events (and state properties and events.) For example, it is common for there to be one behavior pattern when a particular input property value is in its nominal subdomain, and a second behavior pattern when that property is outside its nominal subdomain. The property definition could itself incorporate this dichotomy of occurrences by defining two subdomains of the property occurrences, for example *subdom1* and *subdom2*. Then, in the action specifying the behavior to occur when the input property is in *subdom1*, the guard condition can reference the defined partition of the property values as a unary constraint on the property. The reference identifies the constraint allowing the behavior without having to explicitly cite the property value limits to be satisfied.

6.5 Composition of unit behavior—Build structures

The Conceptual Metamodel includes three features with respect to the composition of unit behavior: build structure, design encapsulation, and obligation dependence. These features of a behavior model constitute a build specification and provide a basis for the behavioral analysis of a system mechanism model. The build specification identifies a composition structure of interacting units, whose individual transformational capabilities are aggregated through coordination of material, energy, and information flows to produce the observable behaviors of the assembly.

6.5.1 Build structures

The notion underlying build structure is essentially the reverse of the notion in 6.1.1, 6.1.3, and Figure 4(b) concerning unit boundary and unit-to-unit interfaces. In that discussion, the oneness of a single named interaction (and its attendant phenomenological basis) is split into separate views from the individual perspectives of the two units sharing the interaction. This introduces a port on each unit to represent its interface with the other, and the (possible) re-labeling of the same interaction instance when talking about the interaction within the context of one unit.

While 6.1.1 is focused on the identification of a boundary segregating the subject system from its environment, the same considerations are used when partitioning any particular unit (including the subject system) into separate design units. The segregation requires consideration of interaction phenomena, as well as the identification of interactions and the identification of conjugate ports to account for the interactions shared between two units. From a top-down design perspective, such successive decompositions result in the definition of perfectly matched interfaces at each unit-to-unit interface. From a bottom-up implementation perspective, adaptation and reuse of existing units could introduce differences between conjugate ports; these differences need to be bounded by compatibility constraints to ensure the behavior of the assembly is not affected by them.

Build structure addresses the accounting of ports and interactions across interfaces. Figure 8 illustrates a build structure of units *U1* and *U2* that are intended to work together. A build structure is the physical pattern by which units share interactions through those ports that constitute the two sides of an interface. In the illustration, Ports *U1.P2* and *U2.P2* are interconnected (mated together) to indicate that the output interaction *Jq* of *U1* affects the behavior of *U2* and that the output interaction *Jp* of *U2* affects the behavior of *U1*.

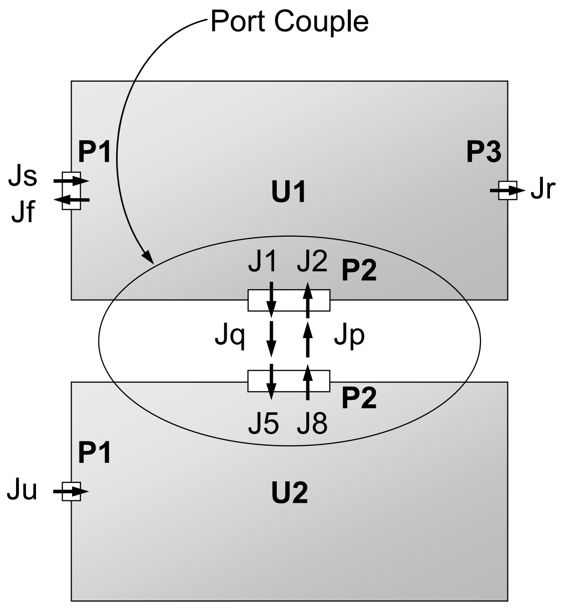


Figure 8—Matching interactions between units through a port couple

The mating of ports *U1.P2* and *U2.P2* is called a *port couple.* For two ports to be identical matches, all output interactions that can occur at one member of the couple would match all input interactions that can occur at the other member of the couple, and vice versa. For such identical matches, the ports are a *perfect* pair of conjugate ports. (Mated ports need not be identical matches, but by satisfying certain compatibility criteria, they are a *compatible* pair of conjugate ports.)

A build specification records a build structure. It identifies the various connector and component units in the assembly and specifies all of the port couples that interconnect them. Through these interconnections, behaviors of the individual units are composed into the net behavior of the assembly.

Though interconnections are shown graphically in the figure as interaction arrows linking the mated ports, an interconnection is not a separate object with its own existence or properties. Specifically, it is not a wire, a pipe, or a communication link between two units or any other form of implementation. Rather, an interconnection in this reference model is no more than a statement of interface identity or compatibility between two units (see 6.1).

It is common for various unit behavior specifications to be developed independently, with possibly different understandings of shared ports and interactions. In addition, units of proven capability can be used in multiple product designs. Two design information management problems arise from this independent development—one syntactic and one semantic.

The syntactic problem is that port interactions that are physically identical are labeled with different names in different specifications. For example, Figure 8 indicates that interaction *Jq* is called *J1* in the specification for unit *U1* and it is called *J5* in the specification for unit *U2*. Conversely, port interactions that are physically distinct may be labeled with the same name.

The build specification provides for the identification of such interaction name aliasing between peer specifications. Thus, the description of a port couple has two parts.

— First, the port couple is defined by enumerating the pair of ports that are to be conjugate to one another. For the example in Figure 8, the following port couple is declared:

Port Couple	*U1.P2, U2.P2*	*// two names for U1-U2 interface*

— Second, the output interactions at each port are matched to the input interactions at the conjugate port. The intended correspondence between interaction names is defined by enumerating these names as interaction aliases. For the example, the following interaction aliases are enumerated:

Interaction Alias	*U1.P2>J1, U2.P2<J5*	*// two specification names for Jq*
Interaction Alias	*U1.P2<J2, U2.P2>J8*	*// two specification names for Jp*

Each statement of an interaction alias in a port couple definition constitutes a kind of compatibility claim, a contradirectional compatibility claim. This claim states a necessary compatibility between an output defined at one port and an input defined at another port. The name *contradirectional* acknowledges that the definitions have opposite directional sense, and this pairing is what properly links the behaviors of the two units. For an assembly to be operationally robust and consistent, all contradirectional compatibility claims must be verified.

As long as all interaction occurrences for *U1.P2>J1* match all interaction occurrences for *U2.P2<J5*, and similarly for the other pair, this identifies a *perfect* pair of conjugate ports.

The semantic problem exists when two ports declared to be conjugate ports are not a *perfect* pair of conjugate ports. As an example, identifying port *U1.P2* and port *U2.P2* as a couple in the assembly in Figure 8 does not require that those two ports be identical in definition or that the output interaction specifications from one be identical with the input interaction specifications at the other. The syntactic definitions of the ports in terms of input and output interactions may be different, or their syntactic definitions may be the same but the definitions of the enumerated interactions may be different.

From the standpoint of successful composition of behaviors and the tracing of causal and affective relationships, the criteria for interconnection are weaker than equality. To couple ports that are not a *perfect* pair of conjugate ports requires the satisfaction of specific compatibility criteria.

— For interactions that could be outputs from one port but do not appear as inputs at the mating port, it is required that no behaviors of the source unit that could cause those interactions can be elicited in the context of this assembly.

— For interactions that are outputs from one port and inputs at the mating port, it is required that interaction occurrences that can be produced by the interaction source unit are a subset of all interaction occurrences that are allowed at by the interaction destination unit.

These criteria require analysis of the individual units for their satisfaction. If, in the example, an output interaction *J3* could be produced by some actions of unit *U1* through port *P2*, it would have to be determined that the stimulus *Js* to *U1* would not elicit those actions. Similarly, if *U2.P2<J5* could accept a limited range of some property of *Jq*, it would have to be determined that those property values in *U1.P2>J1* would not exceed that range for any actions that would be elicited by the stimulus *Js* to *U1*.

6.5.2 Design encapsulation

Encapsulation is the notion of behavioral equivalence through architectural composition. It is the use of a build structure for relating a specification of the behaviors of an assembly to specifications of the behaviors of the units composing that assembly.

The example assembly in Figure 8 is extended in Figure 9 to illustrate a unit *U* that has three interface ports (*U.P1*, *U.P2*, and *U.P3*) to its environment. Through these ports occur instances of two input interactions *U.P1<Ja* and *U.P3<Jc*, and instances of two output interactions *U.P1>Jb* and *U.P2>Jd*. To define its functional role in that environment, a unit behavior specification catalogs those actions of unit *U* that relate sequences and contents of its output interaction occurrences to sequences and contents of its input interaction occurrences.

The nesting of Figure 8 within the figure for unit *U* suggests that the behaviors of unit *U* may be implemented as a composition of the behaviors of two subunits *U1* and *U2*. In a correct and complete build specification, there is a complete behavioral equivalence between the behaviors specified for unit *U* and the composition of behaviors of subunits *U1* and *U2*. This means that all behavior patterns specified for unit *U* are derivable from the individual behavior patterns of subunits *U1* and *U2* when linked through their interconnected ports *U1.P2* and *U2.P2*. The two interactions through port *U.P1* are intended to be accepted and produced through port *U1.P1* on subunit *U1*. The interaction through port *U.P2* is intended to be produced through port *U1.P3* and the interaction through port *U.P3* are intended to be produced through port *U2.P1*.

A conjugate relationship exists between ports *U1.P2* and *U2.P2* in the port couple because the ports represent opposite sides of one interface. The pair-wise port relationships for *U.P1*, *U.P2*, and *U.P3* are quite different. These pairings identify two ports describing the same side of one interface, but at different levels of assembly. Two port names such as *U.P1* and *U1.P1* are related as port aliases. The build specification provides for the identification of such port name aliasing between unit specifications at adjacent levels of assembly. It is necessary that both ports paired in a port alias provide descriptions that are compatible, if not actually identical.

In a top-down design decomposition, the port at the lower level would be specified to be identical with the higher level port. Recording the alias relationship between two such ports provides a simple accountability trace for the design. When all interactions that can occur at one member of the alias match all interactions that can occur at the other member of the alias, the ports are a *perfect* pair of alias ports, providing two port names (in different unit specifications) for a single interface definition.

Aliased ports need not be identical matches. When a preexisting unit is incorporated into a system structure as a lower level unit, its port definitions may differ from the higher-level unit's design specifications for the corresponding ports. Though they are different, the corresponding higher level and lower level ports are a *compatible* alias pair if they satisfy the compatibility criteria described as follows.

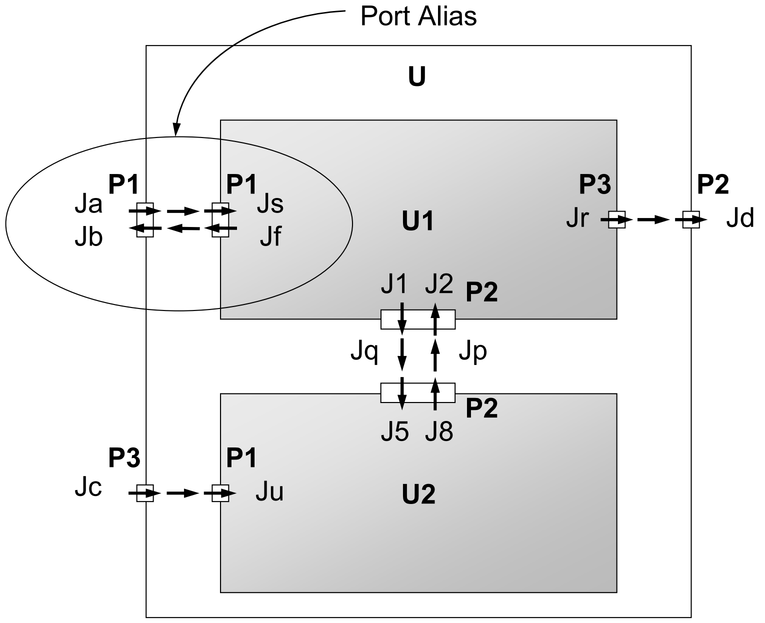


Figure 9—Matching interactions across levels of structure through a port alias

The description of a port alias has two parts.

— First, the port alias is defined by enumerating the port at the lower assembly level that is intended as an alias for the port at the higher level. In the example in Figure 9, the following port alias is declared:

Port Alias	*U.P1, U1.P1*	*// two names for (Ja,Jb) interface*

— Second, the interactions at one port are matched to the interactions at the alias port. The intended correspondence between interaction names is defined by enumerating these names as interaction aliases. For the example, the following interaction aliases are enumerated:

Interaction Alias	*U.P1<Ja, U1.P1<Js*	*// two names for input*
Interaction Alias	*U.P1>Jb, U1.P1>Jf*	*// two names for output*

Each statement of an interaction alias in a port alias definition constitutes a second kind of compatibility claim, a codirectional compatibility claim. The claim states a necessary compatibility between an output from one port and an output from another port. (Alternatively, it states a necessary compatibility between an input into one port and an input into another port.) The name *codirectional* acknowledges that the two definitions have the same directional sense, and this pairing is what properly aliases the interactions at these two ports. For an assembly to be operationally robust and consistent, all codirectional compatibility claims must be verified.

When two ports are not perfect aliases, behavior equivalence between unit *U* and the composition of subunits *U1* and *U2* is still possible as long as the ports are compatible aliases. There are two kinds of interaction compatibility criteria for alias ports.

— For input interaction compatibility, the interaction occurrences at the upper level port (e.g., port *U.P1*) are a subset of all interaction occurrences accepted at the lower level port (e.g., port *U1.P1.*). The lower level port may accept a more extensive set of occurrences of any one interaction kind and may even accept occurrences of other interaction kinds.

— For output interaction compatibility, the interaction occurrences at the upper level port (e.g., port *U.P2*) are a subset of all interaction occurrences produced at the lower level port (e.g., port *U1.P3*).

Also, compatibility requires that the lower level port produces at least the set of occurrences expected at the upper level port for each interaction kind specified at that port.

These interaction compatibility criteria are supplemented with interaction and behavior constraints to ensure behavior compatibility between unit *U* and subunits *U1* and *U2*.

In a correct and complete build specification, the port alias between *U.P1* and *U1.P1* is valid if the following criteria are satisfied. To be input compatible, all input interactions accepted by port *U.P1* can also be accepted by port *U1.P1*. To be output compatible, the outputs produced at port *U1.P1* can not include any that are not included in the specification of port *U.P1*.

Though *U1* may have numerous behaviors that can be solicited by particular interactions at *U1.P1*, the only behaviors of *U1* that can be solicited in this context (input interactions accepted at *U.P1*) are those that do not produce any outputs at *U1.P1* that are not acceptable as outputs from *U.P1*. Compatibility in this assembly configuration means there are identified constraints to ensure that the unacceptable interaction occurrences do not occur. Each statement of an interaction that cannot occur at a port constitutes a third kind of compatibility claim, a denial compatibility claim. The claim states that possible occurrences of a specified lower level output interaction cannot be allowed to occur in this particular assembly configuration. The name *denial* acknowledges that this interaction is not paired with any interaction. at the upper level alias port. For an assembly to be operationally robust and consistent, all denial compatibility claims must be verified. Such verification involves an analysis of peer behaviors and peer-to-peer interactions at the upper level to ensure that stimuli leading to the denied interaction are not available to the lower level unit.

Each of the subunits in the assembly has its peculiar behaviors cataloged in its own unit behavior specification. Behaviors of interconnected units interact with each other as well as with the environment outside the composition. Design verification establishes the equivalence between the specification of unit *U* behaviors and the composition of the subunit behaviors through the architectural configuration.

This verification links the causal and affective relationships in the subunits through port connections within the assembly to produce a set of net causal and affective relationships for the assembly as a whole. These net relationships should be analyzed for comparison with the relationships specified for unit *U*.

A top-down verification of equivalence proceeds through the catalog of actions for unit *U*. For each action in turn, the action selection criteria are distributed appropriately to the assembly subunits and the prediction procedure in 6.5 is applied in each subunit to generate the corresponding results. For subunit results that are linked to other subunits in the assembly, the prediction procedure is applied again to the affected subunits. This iterative application continues until there are no more subunit action results affecting other subunits. The subunit action results are aggregated and compared with the result of the unit *U* action whose selection criteria were used in initiate the chained prediction cycles.

Unit design decompositions are not unique. For any unit behavior specification, there are many reasonable design concepts based on identification of different subunits, different behavior allocations to those subunits, and different interconnection configurations. While the selection of design concept attempts to optimize various decision criteria while satisfying specified measures of performance and effectiveness, there is one absolute constraint: in order for a unit decomposition to be logically consistent, the composition of subunit behaviors, when exhibited in the assembly configuration, must produce all behaviors specified for the unit.

A verification analysis provides evidence that the composition of defined subunits is capable of providing all actions specified for unit *U*. It cannot provide evidence that the composition is not capable of exhibiting other behaviors not specified for unit *U*. That is a difficult task, but behavior specification catalogs of actions for the unit and subunits provide the information required for it.

The implementation of a behavior specification comprises a collection of specific subunits, their individual behaviors, and their configuration of interconnections. While the configuration of particular units is often static, it need not be so. A noteworthy feature of design decomposition evinced in some units is a dynamic assembly configuration of its design subunits. The unit may be realized with subunits that actually engage or disengage at various times. In fact, a reconfigurable assembly of a fixed collection of design subunits having fixed functionality can be used to provide a wide variety of unit functionality. Alternatively, a reconfigurable assembly may provide additional reliability, safety, or performance for a fixed functionality.

In such multi-context cases, the development of the behavior specifications for a subunit is more complex, as some ports and interactions might be used in multiple configurations and some might be used in only one configuration. In a complete and correct behavior specification, all behaviors must be included in their designed capability. The complete behavior specification for each subunit should be verified in every configuration for the unit behaviors in which it participates. This design approach is one way to implement radical differences in behavior states of the unit.

6.5.3 Obligation dependence

Obligation is the notion of collaboration between units in the provision of a particular behavior from their assembly. The pattern for this collaboration is commonly called *client-server*. This is a common design pattern, particularly in information systems. It is illustrated in the build specification in Figure 8 in which unit *U1* plays the client role and unit *U2* plays the server role.

The essential element of the client-server pattern is that the client unit is not independently capable of generating its intended behavior without the assistance of the server unit.

One application example (an information system) fitting the illustration in Figure 8 is the following:

— Interaction *Ju* is a periodic update of information from some external source. Unit *U2* receives this update and keeps the most recent information until the next update arrives. Unit *U1* interacts with a user who issues interaction *Js* to request delivery of most recent information. Unit *U1* does not have this and uses *Jq* to solicit *U2* to retrieve a copy. Unit *U1* receives the copy through *Jp* and delivers it to the user in interaction *Jr*.

Another application example (a physical system) fitting the illustration in Figure 8 is the following:

— Interaction *Ju* is a periodic batch transfer of fuel to an automobile. Unit *U2* is a fuel delivery subsystem comprising a tank, piping, and fuel injection pump. Unit *U1* is the engine with fuel injectors, combustion chambers, and crankshaft. Accelerator position *Js* requests power from unit *U1* and torque is delivered through interaction *Jr*. To generate this power, unit *U1* requires varying fuel through *Jp* and solicits this with *Jq*.

In both cases, unit *U1* is designed explicitly to impose an obligation on its environment. In the first case, the obligation is satisfied by unit *U2* receiving, keeping, and providing the latest information. In the second case, the obligation is satisfied by unit *U2* receiving, keeping, and providing oil as needed (and while it is available) for use by the furnace. In both cases, unit *U1* fails to provide its specified behavior if unit *U2* fails to satisfy the obligation.

There are two distinct interaction patterns around unit *U1*.

— In the service elicitation pattern, the unit first receives an occurrence of the elicitation stimulus *Js*, and it generates later the corresponding occurrence of the satisfaction response *Jr*. The interaction occurrence pattern for normal service is for input to precede output.

— In the obligation solicitation pattern, the unit first generates an occurrence of the solicitation request *Jq*, with an expectation to receive a later occurrence of the solicitation reply *Jp*. The interaction occurrence pattern for an obligation is for output to precede input.

In addition to the input/output pattern reversal of an obligation, there is a required temporal nesting of the service and obligation occurrences. Affective dependency requires the following ordering of the times of interaction occurrence.

Js precedes *Jq*   *Jq* precedes *Jp*   *Jp* precedes *Jr*

Obligation solicitation does not occur before the service request, obligation satisfaction does not occur before the obligation solicitation, and service satisfaction does not occur before the obligation satisfaction.

6.6 Prediction of unit behavior

The behavior specification comprises a catalog of all possible actions of a particular unit. These actions are "units of behavior" for the unit. The following inference procedure uses the actions in this catalog to predict the results of a sequence of unit behavior occurrences.

At some point on an evolutionary lifeline the unit will have accumulated a particular history of interaction sequences and interaction content, characterized by a current behavior and property state. At that point, one or more actions in the catalog would be possible for the unit. Depending on the next input stimulus to occur, and on other available inputs, one of the action selections would be TRUE and the action statement implies the unit would produce the specified action conclusion. In so doing, the unit is at a new point on its evolutionary lifeline, with an updated state, and the predictive cycle can be applied again.

The catalog of actions is a multiverse description from which all possible unit lifelines could be constructed in accord with the possible stimulus interactions received from the environment. For deterministic lifeline construction, it is necessary that the action selections be exhaustive and mutually exclusive and that the conclusions be sufficiently comprehensive in specifying changes to interaction history (behavior and property states) that the inference procedure chain remains viable.

Given a behavior specification, this inference procedure is useful for developing answers about intended unit behavior for a broad range of questions such as the following. (Placeholders like ⟨*this*⟩ stand for particulars that would be meaningful in an application domain.)

a) Questions about specification completeness
    1) Does the specification include all controllability and observability properties of the unit?
    2) Can all intended evolutionary paths of the unit be constructed from the catalog of actions?
b) Questions about behavior possibilities
    1) Can the unit do ⟨*whatever*⟩ in ⟨*circumstance*⟩ ?
    2) In what ⟨*circumstance*⟩ can the unit do ⟨*whatever*⟩ ?
c) Questions about "what if" or "how" scenarios
    1) What would be the result of some sequence of particular stimuli with certain inputs?
    2) What sequence of particular stimuli and inputs would produce ⟨*desired effect*⟩ ?
d) Questions about unit architecture and design
    1) What internal units, design mechanisms, and operations deliver ⟨*action*⟩ into the unit's environment?
    2) How does a particular design mechanism achieve the action selection specification and produce the action conclusion?
e) Questions about unit verification

1) Are there test cases for exhibiting behavior occurrences for ⟨*action*⟩ ?
2) What is necessary for testing the implementation of ⟨*action*⟩ ?

7. Conceptual Metamodel requirements

In accord with the discussion of metamodels in Clause 5, this clause identifies the elements, attributes, and relationships in the Conceptual Metamodel for which a conforming Data Metamodel shall provide representations and structures of data elements, data values, and expressions.

The elements, attributes, and relationships identified in this clause are distilled from the modeling approach described in Clause 6 and satisfy the stated metamodel requirements presented in Clause 4.

7.1 Requirements for metamodel organization

7.1.1 Organization of metamodel elements

The Conceptual Metamodel provides 13 primary concepts for describing the observable features of the behavior of a realizable object. These concepts are grouped into distinct behavior specification perspectives, as enumerated in Table 1.

Table 1—Behavior specification perspectives

Perspective	Description	Concepts
Identification	The realizable object whose behaviors are modeled with a behavior specification	Unit
Boundary Interfaces	Observable characteristics and interpretations of interaction occurrences at the boundary of a unit	Port Interaction
Behavior Patterns	Causal and affective relationships between interaction occurrences that catalog all controllability and observability behaviors for a unit	Action State Obligation
Base Quantities	Quantifiable characteristics that can be observed, measured, and used to quantify a specification of the boundary interfaces and behavior patterns for a unit	Property Event Condition
Build Structures	Composition relationships relating the interfaces and behaviors of multiple units in an assembly to the interfaces and behaviors of the assembly	Assembly Port Couple Port Alias Interaction Alias

A Data Metamodel conforming to this Conceptual Metamodel shall provide representations and structures of data elements, data values, and expressions that organize and integrate the specification perspectives identified here.

a) A specification of Boundary Interfaces for a particular unit can be independently stored and transferred.
b) A specification of Behavior Patterns for a particular unit can be independently stored and transferred.
c) A specification of Base Quantities for a particular unit can be independently stored and transferred.

d) A specification of Build Structures for a particular unit can be independently stored and transferred.

e) Referential integrity between the perspective specifications can be established.

Boundary Interface concepts are those that create the context for defining observable behavior. They identify a unit by the ports and interactions through which its behaviors are accessible and observable.

Behavior Pattern concepts focus on the unit. They describe the causal and affective relations that are realized among occurrences of interactions around a unit: stimulus-response patterns and input-output dependencies.

Base Quantity concepts are those that exist in space and time and are perceptible by external units (people, other systems) in the environment.

Build Structure concepts focus on compatibility claims for interaction identification and behavior composition. They describe the influence relations between units that are realized by an interaction—its availability, its cause, and its alternate identification between behavior specifications.

Figure 10 illustrates the relationships among the metamodel perspectives.

The relationships in Figure 10 are generalizations of specific relationships that exist among the various primary concepts.

The essence of unit behavior specification is that a unit has a boundary comprising interfaces with all other units with which it shares interactions. The observable behaviors of the unit are exhibited by the interactions at those interfaces. Characteristics of the interfaces and the behaviors are quantified with the base quantities of properties, events, and conditions.

There are two distinct relationship sets on Unit, Build Structure, and Boundary Interfaces. A build structure contains some collection of units that are assembled by connecting boundary interfaces on different units to one another, to allow interactions being generated by one unit's behavior to affect another unit's behavior.

In an assembly, the unit interfaces that are not connected to one another form a set of available interfaces to the assembly; further, the composition of unit behaviors that is accomplished through their connected interfaces forms a set of available behaviors of the assembly. The assembly can therefore be recognized as another unit that has available boundary interfaces that exhibit compounded behavior patterns. This is how a unit has a build structure and a build structure provides interfaces.

Each of these perspectives serves a role in understanding the other perspectives.

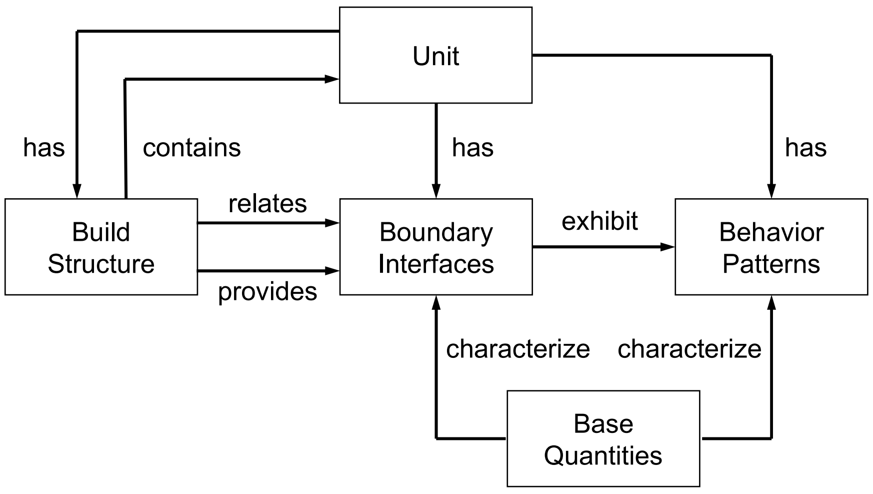
— Boundary Interface elements provide a vocabulary of interactions and properties for defining behaviors.

— Behavior Pattern elements define actions and states in terms of interactions and their properties.

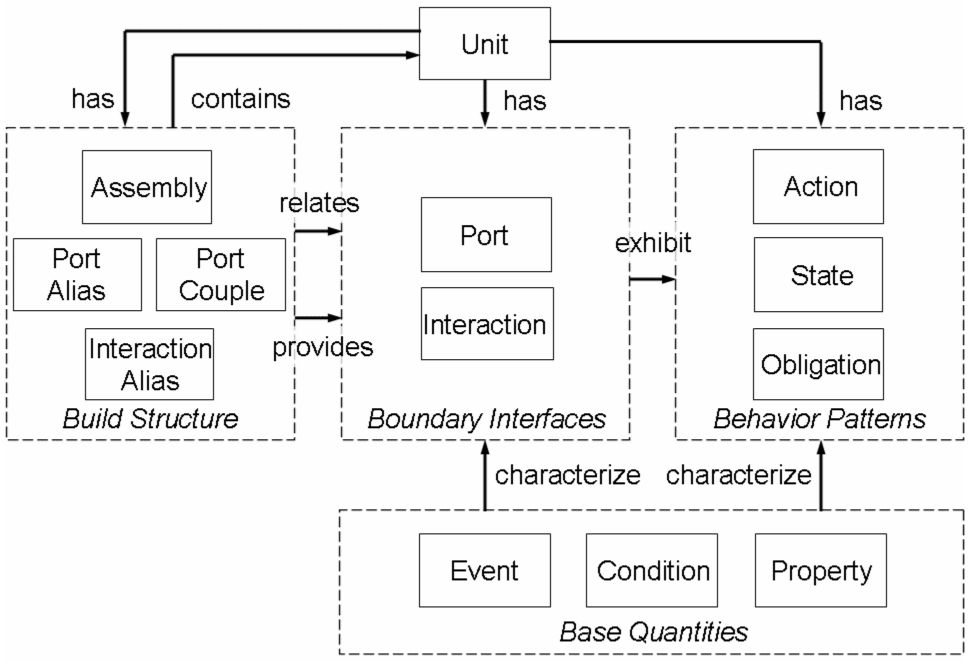
— Basic Quantity elements provide the specification of quantitative observations that relate the model specification of the subject unit to the real world realization of the subject unit.

— Build Structure elements define units and interconnections (each unit with interface and behavior) whose inferred behavior provides the net observable behavior of the assembly of units.

The purpose and role of each Conceptual Metamodel element is summarized in Table 2.



(a)



(b)

Figure 10—Conceptual Metamodel perspectives on behavior metamodel elements

Table 2—Conceptual Metamodel elements grouped by unit behavior model perspective

Perspective	Element	Purpose and role
Identity	Unit	Identifies subject of a behavior model Has defined set of interactions with the environment (via ports) Has defined set of interaction occurrence patterns (as actions and obligations) Creates occurrences of output interactions based on occurrences of current and previous input interactions
Boundary Interfaces	Port	Groups interactions that are shared between two units Represents common phenomena and mechanism by which specified interactions propagate serially into a unit or from a unit
	Interaction	Identifies an observable influence of one unit's behavior on another unit's behavior
Behavior Patterns	Action	Describes a behavior of a unit as a unique pattern of causal and affective relationships between sequences of input and output interaction occurrences
	State	Represents the influence of previous unit behaviors on the unit, providing a context for future unit behaviors Determines what interactions are allowed, what actions patterns are available for solicitation, what dependency future action interactions have on previous action interactions, and what temporal dependency future events have on previous events
	Obligation	Describes a necessary collaboration pattern in which one unit sends as output an interaction to solicit some behavior from another unit and receives as input an interaction produced by that unit
Base Quantities	Property	Describes a measurable phenomenological characteristic of an interaction
	Event	Describes a moment in time at which some perceptible phenomenological change (energy, matter, or information) occurs at a port of the unit
	Condition	Describes a proposition about properties, events, or occurrences and inferences about interactions and actions that may be observed as TRUE or FALSE at a moment along the unit's lifeline Determines a particular lifeline occurrence from other possible occurrences
Build Structures	Assembly	Describes a collection of interconnected units exhibiting a composition of unit behaviors
	Port Couple	Pairs a port of one unit with a port of another (structural peer) unit indicating that all interactions output from one are input to the other, and vice versa
	Port Alias	Identifies a port of one unit with a port of a subunit indicating that all interactions through one are also through the other (used to equate different names for the same port)
	Interaction Alias	Identifies an interaction through one port with an interaction through a port couple or port alias for accounting identity or compatibility of ports in an assembly

The principal causal and affective dependencies in unit behavior are described by the configuration of Conceptual Metamodel elements illustrated in Figure 11.

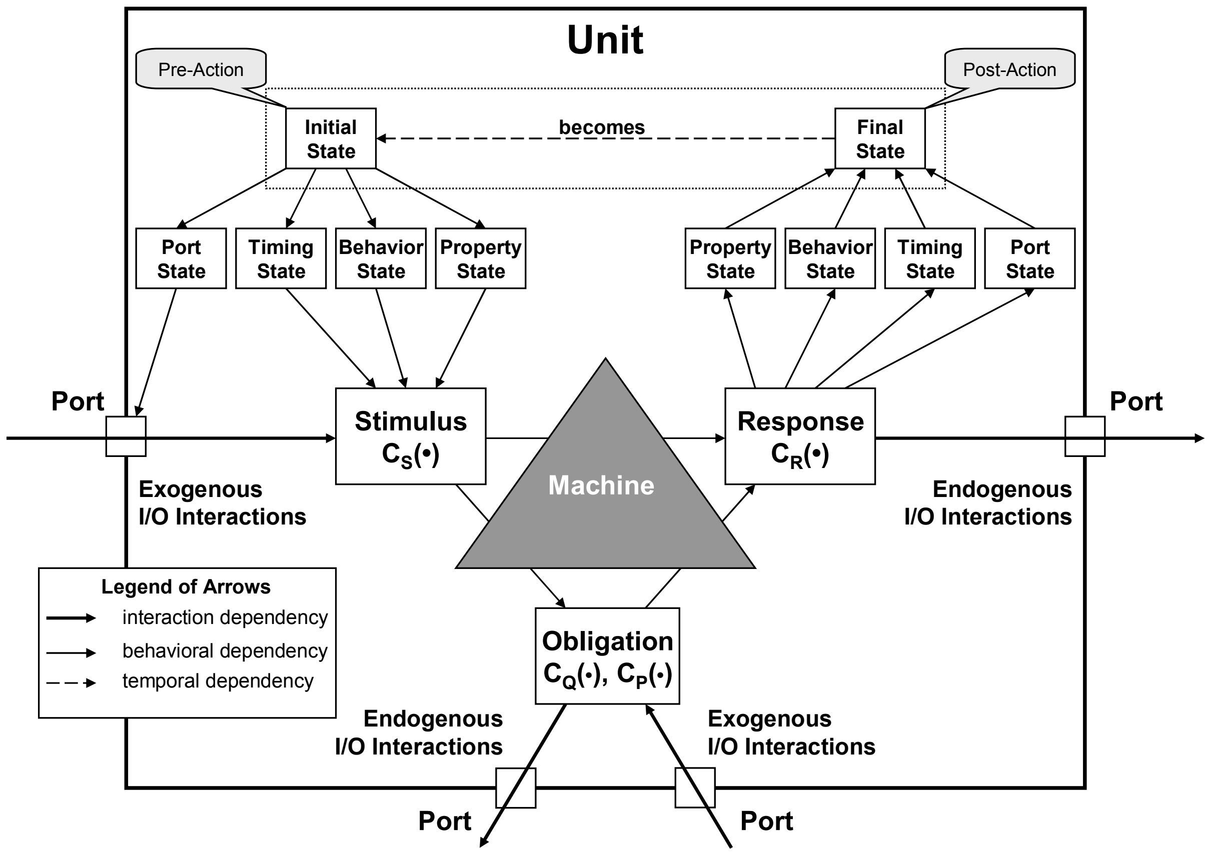


Figure 11—Configuration of metamodel elements describing the causal and affective dependencies of unit behavior

The following interpretation of Figure 11 provides a summary of 6.1 through 6.4.

The subject unit is shown with a boundary containing identified ports through which interactions occur. Exogenous interactions—caused by external units—occur at some ports; these interactions may be either input to or output from the subject unit (I/O). Endogenous interactions—caused by the subject unit—occur at some ports; there interactions may also be inputs or outputs.

In the middle third of the figure, simple causal and functional behavior is described. An exogenous interaction acts as a stimulus to the unit. This stimulus results in the generation of a response that is manifested in an endogenous interaction. The black-box machine realizes this behavior pattern. A precondition $C_S(\bullet)$ may be associated with the recognition of the exogenous interaction as a stimulus. It would identify any constraints or particular properties the interaction must satisfy in order to cause the response. Correspondingly, a postcondition $C_R(\bullet)$ may be associated with the endogenous response interaction. It would identify any guarantees (constraints or relations) the unit makes about resulting interaction when $C_S(\bullet)$ is true. For example, a functional relation would be expressed by $C_S(\bullet)$ constraining the exogenous interaction to be in the domain of the function and $C_R(\bullet)$ guaranteeing the response interaction satisfies the functional relation with the stimulus.

In the top third of the figure are shown the various ways in which behaviors at one time may affect behaviors at later times. The dependence of one behavior on previous behaviors is called *stateful behavior.*

The basis for such stateful behavior comprises four kinds of delayed affective elements. Each of these affects the later behavior in a uniquely observable way.

On the left side of the figure, an initial state with occurrences of the four elements determines what behaviors of the unit are stimulated.

— The observable effect of a boundary state is to indicate whether the unit allows certain exogenous interactions through its ports, that is, whether specific interactions with the unit are available for external units to cause. The boundary state allows or forbids the possibility of certain interactions stimulating any behavior.

— The observable effect of a temporal state is to relate the passage of time within the unit to interaction event times that were previously shared between the unit and its environment. The behavior specification for the unit identifies periodic patterns, prescribed patterns, or single-shot time delays relative to previous events. These patterns and previous events define later internal events (time points) at which the unit may be intended to perform specified behaviors.

— The observable effect of a behavior state is to allow or forbid the performance of certain unit behaviors even when their stimuli are properly identified. The most frequent observation is that different occurrences of exactly the same stimulus result in different behavior responses.

— The observable effect of a property state is to determine current interaction properties as they may depend on previous interaction properties. The property state accounts for the net effects of previous property flows into and out of the unit over the unit's behavior history. Accounting may be cumulative across input and output flows (for conservative property balance) or replicative (for non-conservative property replacement and recall) between input and output flows.

The precondition $C_S(\bullet)$ may determine the concurrent acceptability of the behavior state, property state, boundary state, interaction property, interaction event, or internal event to qualify as a stimulus for some unit behavior. If so, the black-box machine realizes an occurrence of the defined behavior pattern, possibly generating endogenous interactions and establishing a final state that can affect later behavior occurrences.

This final state on the right side of the figure has (possibly different) occurrences of the same four state elements as affected by the behavior occurrence. The final state occurrences may or may not be changed by the behavior occurrence. Absent any identified changes specified by a behavior pattern, the final state elements are duplicates of the initial state elements. Identified changes could include the following:

— The boundary state may allow or forbid different exogenous interactions at different ports.

— The temporal state may be missing some internal event times present in the initial temporal state, and it may have new event times that were not present in the initial temporal state.

— The behavior state may allow behaviors forbidden in the initial behavior state, and it may forbid behaviors allowed in the initial behavior state.

— The property state may have different conservative property balances or different non-conservative property replacement.

The postcondition $C_R(\bullet)$ may identify not only guarantees the unit makes about resulting interactions, but also guarantees about the final boundary state, the final temporal state, the final behavior state, and the final property state. Such guarantees may be absolute or they may be relative to the initial state elements and interactions occurring as part of the behavior occurrence.

The temporal dependency labeled "becomes" at the top of the figure represents the dynamic advancement of the unit through time so that the final state for one behavior (Post-Action) becomes the initial state for a successor behavior (Pre-Action).

In the bottom third of the figure, obligation coordination is incorporated into the causal behavior. When a stimulated behavior requires the involvement of other units in the environment to produce the expected response, the behavior includes the generation of an endogenous interaction through some port of the subject unit to request performance of the obligation by an external unit. The satisfaction of the obligation results in an exogenous interaction reply from an external unit through some port of the subject unit to provide content or timing with which the black-box machine completes the response generation.

Characteristics of the endogenous interaction(s) soliciting the obligation(s) may be guaranteed in request conditions $C_Q(\bullet)$ to satisfy certain functional constraints or relations. Those guarantees may be absolute or they may be relative to the initial state elements and interactions occurring as part of the behavior occurrence. Similarly, the exogenous interaction response from the obligation may be required to satisfy constraints or relations identified in a reply condition $C_P(\bullet)$ for the unit to generate the specified responses. Those requirements also may be absolute or they may be relative to the initial state elements and interactions occurring as part of the behavior occurrence.

Figure 11 provides a guide to the predictive interpretation of the Conceptual Metamodel for behavior specification. The detailed requirements for the elements in this declarative metamodel are contained in 7.2 through 7.5.

7.1.2 Specification of metamodel elements

The specifications of the Conceptual Metamodel are given in the collection of textual statements in 7.2 through 7.4. A set of standardized expressions are used to state these specifications. The collection of these standardized expressions is called a *meta-metamodel,* as they are the rules for stating the Conceptual Metamodel. A description of these standardized expressions follows.

The conceptual elements are labeled with capitalized nouns; adjective-noun pairs are joined with an underscore "_" into a single term. In a behavior specification for a particular unit, each named conceptual element has a domain of distinct, named referents in the real or imagined unit.

The referents for the named conceptual elements are generally of two kinds: static objects that have a constant physical or definitional existence, or dynamic objects that have a time-dependence in their existence. Referents may be indicated as individual objects or as collections of objects.

Each sentence in the Conceptual Metamodel states a relation between particular conceptual elements. In the behavior specification for a particular unit, each stated relation in the Conceptual Metamodel has a relation instance for each named referent in the domain of the participating conceptual elements.

Principal relations are aggregation relations, subtype relations, subset (subdomain) selections, or functional dependencies.

Some sentences are remarks concerning definition, usage, or relationship between the conceptual element and its referent in a real-world unit.

For comprehensiveness of a behavior specification for a particular unit, the conceptual element domains must include all possible referent instances. For example, an external unit always controls the actual property values in an input interaction, but it does not always guarantee that the property value will be in the subdomain of values expected or desired by the subject unit. To account for those interaction occurrences when the property values are out of range, the behavior specification must first identify those undesirable or unusable values that are possible as being members of some subdomain, and must then define alternative action patterns that are to be exhibited for such occurrences.

7.1.3 Implementation of metamodel elements

A Data Metamodel is an enabling system for the storage and transfer of behavior specifications.

A Data Metamodel conforming to this Conceptual Metamodel shall provide representations and structures of data elements, data values, and expressions for ensuring consistency and meeting the model application requirements in Clause 4 and Clause 6. Specifically:

a) The Data Metamodel provides representations and structures for defining a Type_Model for the subject Unit. The Type_Model provides intensive definitions of the concept instances used as the model elements in an Instance_Model of a Unit. Type definitions can be referenced by multiple Instance definitions so as to guarantee appropriate identities of Instance definition. The Type_Model is an intensive model of the Unit's behavioral features.

b) The Data Metamodel provides representations and structures for defining an Instance_Model for the subject Unit. The Instance_Model contains named Instances of concepts that represent designated observables and abstractions of a real or imagined subject Unit. Named Instances of concepts are the vocabulary by which all realizations of Conceptual Metamodel relationships are expressed. The Instance_Model is an extensive model of the Unit's behavior.

c) The Data Metamodel provides representations and structures for defining an Occurrence_Model for the subject Unit. The Occurrence_Model represents a segment of a possible lifeline of the Unit. This model includes the following elements:

    1) Appropriate occurrences of State, exogenous Interactions, and Events selected to represent the desired lifeline environment of the Unit.

    2) A possible lifeline containing a time-annotated sequence of interaction and Action occurrences representing one of the Unit's possible behaviors.

7.2 Requirements for representing Boundary Interfaces

Boundary Interfaces (see 6.1 and 6.2) specify the boundary that segregates the unit from its environment.

A Data Metamodel conforming to this Conceptual Metamodel shall provide representations and structures of data elements, data values, and expressions that enable the storage and transfer of information describing instances and occurrences of the conceptual elements and relationships identified in the following list:

a) The Boundary Interfaces of a named Unit are described by a collection of one or more named Ports. Each Port represents an interface with another unit. Ports are named uniquely within the scope of a unit specification.

b) Multiple named Interactions may be associated with a particular named Port. Interactions are named uniquely within the scope of a Unit specification.

c) A named Interaction at a Port has two directional subtypes with respect to the subject unit: Input_Interaction and Output_Interaction.

d) A named Interaction at a port has two occurrence control subtypes: Endogenous_Interaction (controlled by the subject Unit) and Exogenous_Interaction (controlled by the external unit with which it is shared.)

e) Depending on the time granularity of the behavior description, an occurrence of a named interaction has two duration subtypes: Point_Interaction (happens at a "point" in time; "point" in time is any time interval smaller than the time granularity) and Extended_Interaction (happens through an interval of time).

f) A Unit may or may not have an Availability_Control over an Exogenous_Interaction.

g) Interaction occurrence has a Begin_Event and an End_Event whose difference is the Interaction_Duration.

h) Interaction occurrence may or may not have Interaction_Content that is relevant to the description of Unit behavior.

i) Interaction_Content has two composition subtypes: Simple_Content and Structured_Content.

   1) Simple_Content comprises one or more Properties. Properties of any of the phenomenological subtypes (matter, energy, information) may be included as appropriate to model Interaction_Content.

   2) Structured_Content comprises component Interaction occurrences in a composition structure.

j) Structured_Content has two structural subtypes: Sequential_Structured_Content and Concurrent_Structured_Content.

   1) Sequential_Structured_Content comprises one or more component Interactions whose occurrences have begin and end events in a sequence of non-overlapping intervals.

   2) Concurrent_Structured_Content comprises one or more component Interactions whose occurrences all have the same begin event and the same end event.

k) The set of all possible occurrences for an Interaction can be partitioned by conditions into two or more named Interaction_Subdomains depending on observable characteristics of either the Interaction_Duration or the Interaction_Content.

   1) With respect to Interaction_Duration, interaction occurrences can be partitioned by Conditions into subsets (subdomains) of different temporal extent.

   2) With respect to Interaction_Content, Simple_Content can be partitioned by Conditions into subsets (subdomains) of different Property values for individual Properties or subsets of different combinations of Property values for multiple Properties.

   3) With respect to Interaction_Content, Structured_Content can be partitioned by Conditions into subsets (subdomains) of different component Interaction_Subdomains or subsets of different relationships among the component Interactions (relationships among contents or relationships among occurrence events).

NOTE 1—Observables of interactions are specified with conditions. Conditions on properties may determine whether those properties satisfy certain criteria. Conditions on events may determine if interactions are too short or too long, or if they are occurring too closely or not closely enough, etc.

NOTE 2—The metamodel uses conditions to partition possible interaction occurrences into those for which the condition is TRUE and those for which it is FALSE. These discriminations allow the association of different behavior patterns with the different subsets. (These subsets are similar to "equivalence classes.")

NOTE 3—Conditions on time-varying properties define events as the time(s) at which the condition changes from TRUE to FALSE or from FALSE to TRUE.

7.3 Requirements for representing Behavior Patterns

Behavior Patterns (see 6.3 and 6.4) specify causal and affective relationships among interaction instances that are made true by the unit as it evolves along its lifeline.

A Data Metamodel conforming to this Conceptual Metamodel shall provide representations and structures of data elements, data values, and expressions that enable the storage and transfer of information describing instances and occurrences of the conceptual elements and relationships identified in the following list:

a) The Behavior Patterns of a named Unit are described by a collection of one or more named Actions. Each Action represents a distinct pattern of causal and affective relationships that the Unit exhibits in its Interactions with other units in its environment. Actions are named uniquely within the scope of a unit specification. The solicitation of an Action can depend on the history of previous Actions performed by the unit. An Action is itself part of the Unit history for later actions.

b) The historical dependency of future Actions of a Unit on past Actions of the Unit is described by a State of the Unit. The State accounts for effects on the Unit by previous behaviors and the dependency of later behaviors on those effects. The State evolves along the Unit lifeline, incorporating the effects of behaviors on the unit and bounding the solicitation and results of later behaviors.

c) The State of the Unit comprises the following parts: a Boundary_State, a Behavior_State, a Property_State, and a Temporal_State.

   1) The Boundary_State is the collection of all Availability_Controls defined for Exogenous_Interactions. A Boundary_State_Occurrence is a particular pattern of Exogenous_Interactions that are available at that point on the Unit's lifeline.

   2) The Behavior_State is a collection of Action_Sets that identify alternative subsets of Unit behaviors that the Unit offers to its environment at various times along its lifeline. This is a partitioning pattern in the Unit specification that distinguishes behaviors that are not possible by the Unit under certain circumstances. A Behavior_State_Occurrence is the identification of the particular Action_Set that describes available behaviors at a particular time on the Unit's lifeline.

      i) An Action_Set identifies a subset of Actions that can be solicited at various points along the unit lifeline. Actions that are not members of an Action_Set are not possible when the Action_Set is the Behavior_State_Occurrence.

   3) The Property_State comprises a collection of zero or more named internal Properties. These are inferred properties that can be affected by Interactions with the Unit and that can, in turn, affect later Interactions of the Unit. These Properties account for the observable effects that past Actions can exert on future Actions. A Property_State_Occurrence is the set of Property values at a particular time on the Unit's lifeline.

   4) The Temporal_State comprises a collection of Events from whose occurrences elapsed times are reckoned along the Unit lifeline. The Temporal_State comprises the Interval_Mark_Event definitions for the Unit. The Events in this collection are those whose occurrences are subjects of propositions that relate these Events to other Events. The Temporal_State represents meaningful timings and intervals for which the Unit is responsible.

d) An Action of the Unit comprises the following parts: an Action_Stimulus (to describe the cause of a behavior), an Action_Response (to describe the effects of a behavior), and optionally an Action_Obligation (to describe any collaborating Interaction exchanges for obtaining services from external units for providing the Action_Response). The Action describes the causal and affective relationship between occurrences of a defined Action_Stimulus and occurrences of a defined Action_Response. The Action further identifies the dependencies of this relationship on services acquired through interaction with external units.

e) An Action_Stimulus comprises the following parts: an Initiation_Event, a Termination_Event (optional), zero or more Exogenous_Interactions, the Initial_State of the Unit, and a Precondition. This combination of elements is unique.

   1) The Initiation_Event is an Event whose occurrence triggers the beginning of a behavior occurrence described by the Action.

   2) The Termination_Event is an Event whose occurrence stops the Unit behavior. Different Precondition propositions on the Termination_Event define a differently named Action_Stimulus, and a different Action_Stimulus is associated by a different Action definition to a different Action_Response.

3) The Initial_State is some subset of occurrences of the Unit State. The Boundary_State_Occurrence is one that allows the Exogenous_Interactions that may be identified in the Action_Stimulus. The Behavior_State_Occurrence is an Action_Set that includes the Action. The Property_State_Occurrence satisfies propositions in the Precondition.

4) Not all possible occurrences of the Exogenous_Interaction(s) may be allowed for the Action_Stimulus occurrence. A proposition in the Precondition identifies those that are allowed.

5) The Precondition is a Condition that may constrain any of the other Action_Stimulus parts (or their constituent elements) individually. It may also constrain relationships between these parts. Occurrences of the other Action_Stimulus parts for which the Precondition is TRUE will cause the Unit to exhibit a behavior described by any of the Actions referencing this Action_Stimulus.

f) An Action_Response comprises the following parts: zero or more Endogenous_Interactions, the Final_State of the Unit, and a Postcondition. This combination of elements is unique.

1) The Final_State is some subset of occurrences of the Unit State. The resulting Boundary_State_Occurrence is specified by a proposition in the Postcondition. The resulting Behavior_State_Occurrence is specified by a proposition in the Postcondition. The Property_State_Occurrence satisfies propositions in the Postcondition. Properties in the final Property_State may be functionally dependent on properties in the initial Property_State, as well as properties of the Exogenous_Interactions and the Endogenous_Interactions.

2) The Endogenous_Interaction(s) are those that result from the behavior occurrence. Properties in the Endogenous_Interactions may be functionally dependent on properties in the Exogenous_Interactions, as well as properties of the Property_State.

3) The Postcondition is a Condition that may constrain any of the other Action_Response parts (or their constituent elements) individually. It may also constrain relationships between these parts. Occurrences of the other Action_Stimulus parts for which the Precondition is TRUE will cause the Unit to exhibit a behavior described by any of the Actions referencing this Action_Stimulus.

g) The set of all possible occurrences for an Action_Obligation can be partitioned into two or more named Obligation_Subdomains depending on observable characteristics of the Interactions defined in that subdomain.

1) An Obligation_Subdomain has an Obligation_Request (by which the Unit solicits service from an external unit) and an Obligation_Reply (by which the Unit receives the result of the solicited service from the external unit).

i) An Obligation_Request comprises one or more Endogenous_Interactions and a Condition. The Condition may constrain the Endogenous_Interaction occurrences or it may constrain the relationship between the Endogenous_Interaction occurrences and the Action_Stimulus occurrence.

ii) An Obligation_Request comprises one or more Exogenous_Interactions and a Condition. For a particular Obligation_Subdomain, the Condition may constrain the Exogenous_Interaction occurrences or it may constrain the relationship between the Exogenous_Interaction occurrences, the Obligation_Request, and the Action_Stimulus occurrence.

NOTE 1—The conceptual model for State does not require any of the common hierarchical Finite State Model structures for its representation. It is concerned only with the partitioning of all actions into (possibly overlapping) sets that absolutely exclude solicitation of some actions at some times along the unit's lifeline.

NOTE 2—The relationship between State and Action instances is a protocol for evolution along the Unit lifeline. Action describes a capability of the Unit to interact with its environment. Every behavior occurrence may have an effect on the Unit itself, that is, on the State of the Unit. State describes the effect of on the Unit of its history of behavior occurrences. Whatever behavior by the Unit occurs, it occurs in the current State of the Unit.

NOTE 3—The explicit relationship between State and Action is identified in Action_Stimulus and Action_Response elements. With respect to Action, Initial_State identifies those possible contexts along the Unit's lifeline when an occurrence is possible, while Final_State identifies the contexts that are created by that occurrence. With respect to State, Actions identify the possible transitions that can be made from a particular Initial_State to various possible Final_States.

NOTE 4—Boundary_States are consistent with Behavior_States if, for all stimulus interactions made available in a Behavior_State, there are actions defining the unit's responses to them.

NOTE 5—Property State as internal effects of interactions provides time shifting of interaction properties. This is common in data based systems where records of previously submitted data can be created, read, updated, and deleted.

NOTE 6—Internal properties are not constrained to reflect interaction properties, as such. Internal properties may be defined in support of other affective relationships in the Unit's behavior. For example, an internal property that counts occurrences of a particular Action may be used to limit the number of times the Action can occur in sequence along the Unit's lifeline.

7.4 Requirements for representing Base Quantities

Quantified observables specify the measurable characteristics of Interaction occurrences at the Boundary Interface and Behavior Pattern occurrences that relate those interaction occurrences.

A Data Metamodel conforming to this Conceptual Metamodel shall provide representations and structures of data elements, data values, and expressions that enable the storage and transfer of information describing instances and occurrences of the conceptual elements and relationships identified in the following list:

a) The Base Quantities of a named Unit are the collection of Properties, Events, and Conditions used in the specification of boundary interfaces and behavior patterns. Properties quantify phenomenological aspects of a Unit specification. Events quantify temporal aspects of a Unit specification. Conditions bound or relate instances of Properties and Events in a Unit specification.

b) A Property has three phenomenological subtypes: Energy_Property, Material_Property, and Information_Property.

    1) Energy_Property represents observable energy measurement phenomena.

    2) Material_Property represents observable material measurement phenomena.

    3) Information_Property represents useful temporal or spatial patterns in Energy_Properties or Material_Properties that can be maintained at a place and time, that can be propagated from one place and time to another place and time, or that can be transferred across a unit's boundary interfaces.

c) A Property has two subtypes: Conservative_Property and Nonconservative_Property.

    1) A Conservative_Property is one that balances increments or decrements of the Property in Interaction occurrences with a corresponding Property in Property_State occurrences. Energy_Properties and Material_Properties are of this type, but additivity differs for extensive and internsive properties. An Information_Property may be treated as a Conservative_Property by a Unit.

   2) A Nonconservative_Property is one that does not balance increments and decrements of the Property in Interaction occurrences with a corresponding Property in Property_State occurrences. An Information_Property may be treated as a Nonconservative_Property by a Unit.

d) A Property has two variability subtypes: Constant_Property and Variable_Property.
   1) A Constant_Property is one whose Interaction or State occurrence value is constant across the duration of Interaction having the Property.
   2) A Variable_Property is one whose Interaction or State occurrence value changes across the duration of Interaction having the Property.

e) A Property has two composition subtypes: Single_Property and Compound_Property. A Single_Property is one for which a single Property value is derived by observation. A Compound_Property is one for which a collection of Property values is derived by observation.
   1) Compound_Property comprises two or more component Property occurrences in some composition structure.

f) The set of all possible occurrences for a Property can be partitioned into two or more named Property_Subdomains depending on various observable characteristics of the Property.
   1) A Single_Property can be partitioned by Conditions into Single_Property_Subdomains.
      i) Single_Property_Subdomain has a Property_Unit (unit of measure), a Property_Scale (reference magnitude of measurement), and a Property_Value_Set (range of measurement values).
   2) A Compound_Property can be partitioned into Compound_Property_Subdomains. Compound_Property_Subdomain occurrences may differ in allowed combinations of different component Single_Property_Subdomains or in different structures and relationships among the component Properties.

g) A Property_Value_Set has two temporal subtypes: Constant_Value_Set and Function_Value_Set.
   1) A Constant_Value_Set is a collection of constant values. Any occurrence of the Property in a particular Single_Property_Subdomain exhibits one of those values.
   2) A Function_Value_Set is a collection of time-dependent functions. Functions have a domain of values and specified functional properties such as maximum rate of change. Any occurrence of the Property in a particular Single_Property_Subdomain exhibits one of those functions.

h) An Event has two source subtypes: External_Event and Internal_Event.
   1) An External_Event arises from a change in the environment. It is associated with Exogenous_Interactions.
   2) An Internal_Event arises from a change in the Unit. It is associated with Endogenous_Interactions, Property_State changes, or the internal time sense of the Unit.

i) An Event has two composition subtypes: a Single_Event or a Compound_Event. A Single_Event uses a particular type of observation to determine the time of Event occurrence. A Compound_Event uses a particular type of inference to determine time based on Event occurrence.
   1) A Single_Event has three marking subtypes: Occurrence_Mark_Event, Change_Mark_Event, and Interval_Mark_Event. Marking is the observation or prediction of a change or transition at a point in time.
      i) An Occurrence_Mark_Event labels the observation of times when a named Interaction begins or ends.

ii) A Change_Mark_Event labels the observation of times at which a Condition defined on a time-dependent Property (in an Interaction or a Property_State) becomes TRUE.

iii) A Interval_Mark_Event labels the observation of times elapsed from a reference Event. This includes an absolute time or interval of clock time in the environment (when the Unit and the environment have previously shared a synchronizing Interaction Event), a relative interval of time following another Event, and periodic or aperiodic sequences of times following another Event. (Interval_Mark_Events are collectively called the Temporal_State of the Unit.)

2) A Compound_Event has three inference subtypes: First_Event, Last_Event, and Concurrence_Event.

i) A First_Event labels the times inferred by the *firstOf*(•) rule from a collection of named Events.

ii) A Last_Event labels the times inferred by the *lastOf*(•) rule from a collection of named Events.

iii) A Concurrence_Event labels the times inferred by the *concurrenceOf*(•) rule from a collection of named Events.

j) A Condition states a Proposition on one or more subjects that is to be TRUE at a specific time of evaluation. The Proposition is a function expressed in terms of arguments that are specific *instances* of specification elements. The function is evaluated as TRUE or FALSE at a particular point of time along the Unit lifeline for particular *occurrences* of the specified concept instances at that point of time.

k) A Proposition has two composition subtypes: Simple_Proposition and Compound_Proposition.

l) A Simple_Proposition has three subject domain subtypes: Property_Proposition, Event_Proposition, and Concurrence_Proposition.

m) Property_Proposition has two arity subtypes: Property_Restriction (monadic) and Property_Relation (dyadic).

1) A Property_Restriction has one Property instance argument. It defines a constraint that accepts a subset of the possible Property occurrences for which the constraint is TRUE. The constraint may restrict the numerical magnitude, lexical pattern, or functional form of the allowed occurrences of the Property instance.

2) A Property_Relation has two Property instance arguments. It defines a binary relationship between occurrences of the two Properties. It accepts a subset of the possible Property pair occurrences for which the constraint is TRUE. The relationship may relate numerical magnitudes, lexical patterns, or functional forms in the two occurrences.

n) An Event_Proposition has two Event instance arguments. It defines a binary relationship on the elapsed time interval between related occurrences of the two Events. It accepts a subset of the possible durations between the two event times.

o) A Concurrence_Proposition has one Interaction instance or one Action instance argument. As an element in an Action Precondition, its evaluation takes place when the Initiation_Event of the Action occurs on a particular lifeline of the Unit. The proposition is TRUE if there is an existing occurrence of the named argument at the time of proposition evaluation.

p) A Compound_Proposition has four logic subtypes: Negation_Proposition, Conjunction_Proposition, Disjunction_Proposition, and Implication_Proposition.

1) A Negation_Proposition has one Condition instance argument. Its value is derived from the argument occurrence by the truth table for logical NOT.

2) A Conjunction_Proposition has two Condition instance arguments. Its value is derived from the argument occurrences in accord with the truth table for logical AND.

3) A Disjunction_Proposition has two Condition instance arguments. Its value is derived from the argument occurrences in accord with the truth table for logical OR.

4) A Implication_Proposition has two Condition instance arguments. Its value is derived from the argument occurrences in accord with the truth table for logical IMPLICATION.

NOTE 1—A Property_Proposition applied to time-dependent property has *edges* in time at which the proposition changes from FALSE to TRUE. These edges define an event type whose occurrences are those times of transition.

NOTE 2—A Concurrence_Proposition is TRUE when the start event for an occurrence of named Interaction or Action has occurred at the time of proposition evaluation, but the end event for that occurrence has not occurred; at the time of proposition evaluation, the occurrence is ongoing. Concurrence_Propositions may be used in Action Preconditions to control Action concurrency.

7.5 Requirements for representing Build Structures

Build Structures (see 6.5) specify assemblies of interacting units whose shared interactions compound the behaviors of individual units into behaviors of the assembly. In use, an assembly specifies an interacting structure of units and the compatibility criteria that guarantee a valid assembly. However validity of the assembly specification does not imply satisfaction of the criteria and correctness of the behavior composition.

A Data Metamodel conforming to this Conceptual Metamodel shall provide representations and structures of data elements, data values, and expressions that enable the storage and transfer of information describing instances and occurrences of the conceptual elements and relationships identified in the following list:

a) A Build Structures for a named Unit is described as an Assembly of interacting Sub_Units. There may be zero or more different Assemblies that provide the behaviors required of the subject Unit. A Unit in the Assembly is called a Sub_Unit to distinguish it from the Unit to which the Assembly is behaviorally equivalent.

b) An Assembly for a named Unit has the following parts: two or more Sub_Units, one or more Port_Couple(s), and one or more Port Alias(es). A Port_Couple identifies a compatible mating of a Port on one Sub_Unit with a Port on a different Sub_Unit. The collection of Port_Couples define the interconnection configuration of the Sub_Units. A Port_Alias identifies a compatible replacement of a Unit Port with a Sub_Unit Port.

c) A Port_Couple identifies two Ports and a set of one or more Interaction_Aliases. Each Port belong to a different Sub_Unit. The criteria for their mating is described by the enumeration of Contrtadirectional_Claims between Interactions at one Port and Interactions at the other Port.

d) A Port_Alias identifies two Ports and a set of one or more Interaction_Aliases. One Port belongs to the Unit and one Port belongs to a Sub_Unit. The criteria for their equivalence is described by the enumeration of Codirectional_Claims or Denial_Claims between Interactions at one Port and Interactions at the other Port.

e) An Interaction_Alias has three compatibility subtypes: a Contradirectional_Claim (describing compatibility of two Interactions in a Port_Couple), a Codirectional_Claim (describing compatibility of two Interactions in a Port_Alias), and a Denial_Claim (describing a disallowed Interaction in a Port_Alias.

   1) A Contradirectional_Claim has two members: an Output_Interaction at one Port of a Port_Couple and an Input_Interaction at the other Port of a Port_Couple. The compatibility criterion implied by this claim is that the domain of occurrences defined for the Output_Interaction is a subset of the domain of occurrences defined for the Input_Interaction for all defined features of the Input_Interaction.

2) A Codirectional_Claim has two directional subtypes: a Codirectional_Output_Claim and a Codirectional_Input_Claim.

    i) A Codirectional_Output_Claim has two members: an Output_Interaction from the Sub_Unit Port and an Output_Interaction from the Unit Port. The compatibility criterion implied by this claim is that the domain of occurrences defined for the Sub_Unit Output_Interaction does not exceed the domain of occurrences defined for the Unit Output_Interaction for all defined features of the Unit Output_Interaction.

    ii) A Codirectional_Input_Claim has two members: an Input_Interaction to the Unit Port and an Input_Interaction to the Sub_Unit Port. The compatibility criterion implied by this claim is that the domain of occurrences defined for the Unit Input_Interaction does not exceed the domain of occurrences defined for the Sub_Unit Input_Interaction for all defined features of the Sub_Unit Input_Interaction.

3) A Denial_Claim has one member: an Output_Interaction from the Sub_Unit Port. The compatibility criterion implied by this claim is that, for all behaviors of the Sub_Unit that can occur when any behavior of the Unit is triggered, no occurrences of this Output_Interaction can be produced.

Annex A

(informative)

Bibliography

[B1] DoD Architecture Framework (DoDAF), Version 1.5, U.S. Department of Defense (DoD), 23 April 2007.

[B2] IEEE 100™, *The Authoritative Dictionary of IEEE Standards Terms*, Seventh Edition. New York: The Institute of Electrical and Electronics Engineers, Inc.[10, 11]

[B3] IEEE Std 1175™-1991, IEEE Standard Reference Model for Computing System Tool Interconnections.

[B4] IEEE Std 1175.1™-2002 (Reaff 2007), IEEE Guide for CASE Tool Interconnections—Classification and Description.

[B5] IEEE Std 1175.2™-2006, IEEE Recommended Practice for CASE Tool Interconnection—Characterization of Interconnections.

[B6] IEEE Std 1175.3™-2004, IEEE Standard for CASE Tool Interconnections—Reference Model for Specifying Software Behavior.

[B7] Technical Report CMU/SEI-2002-TR-011, CMMI® for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1, Continuous Representation, Carnegie Mellon University (CMU), Software Engineering Institute,[12] http://www.sei.cmu.edu/publications/documents/02.reports/02tr011.html.[13, 14]

[B8] Wadge, W. W., and Ashcroft, E., A., *Lucid, the Dataflow Programming Language*. Academic Press, 1985.

[10] IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08854-1331, USA (http://standards.ieee.org/).
[11] The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.
[12] The CMU Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.
[13] CMMI stands for Capability Maturity Model® Integration.
[14] Capability Maturity Model and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. This information is given for the convenience of users of this standard and does not constitute an endorsement of the IEEE of these products. Equivalent products may be used if they can be shown to lead to the same results.

Annex B

(informative)

Comparison of the system behavior model with other engineering models

In this annex, the focus and content of the system behavior metamodel defined in this standard is compared and contrasted with several other kinds of engineering models used in system development. Each model has a purpose that is served by the choice of descriptive elements in the model, the questions that can be posed to and answered with the model, and the methods by which inferences are made. Many different models are constructed for large and complex systems. Absent a single integrated "model of everything about the system," it is possible that the relationships among the various models of a system might lack coherence, allowing conflicts to arise in the use of these models.

Each of these models can be expressed in a variety of representations. The choice of representation depends on that representation's capability to represent the things and relationships being modeled, the adequacy of inference mechanisms for extracting information from the model, and the ease of creating the model and extracting inferences.

NOTE 1—Although the types of models summarized in this clause are labeled as "system" models, those describing behavior, mechanism, implementation, and verification can be used extensively for assemblies and parts within a system. The need for such uses is discussed in Technical Report CMU/SEI-2002-TR-011 [B7].

NOTE 2—The model descriptions in this clause are conceptual only. They are intended to show the principal purpose and use of some common kinds of models. The list of models is not exhaustive, nor is the exposition of modeling issues to be addressed in each area.

The following models are addressed:

— Stakeholder requirements model
— System use model
— System mechanism model
— System implementation model
— System verification model
— System validation model

B.1 Stakeholder requirements model

A stakeholder requirements model focuses on the application domain in which the system is to operate. It models the effects that occur in the application domain by virtue of the operation of the subject system. This is a two part model in that it models the real world neighborhood of the system from two perspectives. The first part of the model identifies the *given environment* of the subject system. This describes what exists and what is true in the application domain in the absence of any effects of a subject system. The second part of the model identifies the *intended effects* that the subject system is required to make true. This describes the effects that system operation is to have on the given environment.

Stakeholder requirements could also be called *customer requirements* or *user requirements,* as they are requirements on the effects of the system in the stakeholders' world.

Things to be described about the *given environment* of the application domain include the following:

- Active and passive elements in the application domain and the phenomena through which they interact
- Cause and effect dependencies among elements in the application domain through which desired effects can be brought about (and undesired effects can arise)
- Phenomena that could be shared between the application domain and the system that would enable system interaction with the application domain
- Events (the occurrence of changes) that provide a reference for time
- Physical laws, equations of state, and constitutive relations that control or constrain properties of the elements or their interactions
- Social expectations characterizing allowed states of the application domain (safety, reliability, etc.)
- Human background, bias, and limitations in interaction perceptions
- Economic, technologic, and legal constraints to which desired effects in the application domain are subject

Things to be described as the *intended effects* system operation in the application domain include the following:

- New elements and relationships to be established
- New kinds of transformations to be accomplished
- New conditions or constraints to be satisfied
- Delayed effects from previous events and activities on later events and activities
- Improved performance (e.g., throughput, robustness, safety, reliability) of existing transformations
- Reduced failures of existing transformations

Prior to the existence of a system, these are requirements to be met. Following development of a system purporting to comply with the model, these provide one basis for accepting the system (through validation).

A common failure in models of stakeholder requirements is the assumption that the given environment is so well-known as not to need description. Failure to validate this assumption for all stakeholders introduces hidden ambiguities into the interpretation of these requirements. It can be crucial for some application requirements to identify and understand relationships existing in the environment. A common omission in models of the stakeholder requirements is the recognition that the system itself is a new element in the application domain, attended by additional relationships with the application domain to be identified and understood.

B.2 System use model

A system use model focuses on the strategies and procedures by which use of the system through its interactions can be choreographed to bring about desired effects in the application domain. Such a model connects the system behavior model to the stakeholder requirements model by describing how to satisfy the stakeholder requirements through scenarios of interaction with the system (i.e., patterns for invoking or eliciting system behaviors and the system responses that affect the application domain).

System use models may be called *concepts of operation.* They focus on the intended interaction protocols by which all system users and beneficiaries are intended to extract services of direct value from the system.

A complete system use model might identify many desired stimulus-response sequences. In general, however, it will not identify all possible sequences for which the system behavior model would prescribe results.

Prior to the existence of a system, the system use model provides some verifiable linkage between the system behavior model and the stakeholder requirements model. Following acceptance of a system, the system use model provides the basis for user documentation.

B.3 System mechanism model

A system mechanism model focuses on a structural and operational decomposition of the subject system into a collection of architectural elements and element interconnections whose aggregate behavior satisfies a system behavior model.

Depending on the level of detail and corresponding terms of description, system mechanism models may be called *architectural composition* or *design.* Mechanism models describe how parts and assemblies of parts provide particular properties and behaviors in the system structure. A complete mechanism model for a system usually has a hierarchical set of such models. Higher level models identify part assemblies as system elements that are independently managed, specified, implemented, and verified. Lower level models identify parts that are manufactured or acquired.

B.4 System implementation model

A system implementation model focuses on the plans, processes, materials, controls, and resources with which parts and part assemblies are fabricated. The realization data characterizing parts and assemblies provide a wide array of models: physical characteristics, chemical characteristics, electrical characteristics, radiation characteristics, etc.

NOTE—Examples are CAD/CAM models made for hardware, schematics and board layouts made for electronic circuits, component and class models for software, and VHDL and lithographic models for semiconductor devices.

B.5 System verification model

A system verification model focuses on the evidence, to be generated through observations and measurements, that a particular system satisfies its specifications. There are two fundamental aspects of such models: a structure of the evidence that supports adequate inference of specification satisfaction, and a sampling approach for acquiring the evidence. Such evidence normally includes technical reviews of various product models (specifications, designs, and implementations) as well as testing of implemented product.

In the large, the sampling approach should determine what properties and behaviors are to be verified at the part level, the assembly level, and the system level, to verify that the functional and structural features of the system are what are intended. In the small, the sampling approach should address selections of specific part, assembly, and system properties and behaviors to be examined and the contexts in which they are to be exhibited.

The verification model addresses the design of stimuli and stimulation methods, methods for response observation and property measurement, data analysis, and traceability to specifications. Design of appropriate external environment surrogates and specific behavior tests can be based directly on a system behavior model. Sampling methods are selected to satisfy criteria for structural and functional coverage.

The primary questions to be answered by this model pertain to adequacy in detecting discrepancies: the density of samples across relevant domains (structure, function, and environment), and adequacy of the surrogate environment in which testing is conducted.

B.6 System validation model

A system validation model focuses on modeling the evidence to be generated, through observations and measurements, that a particular system fulfills the stakeholder requirements in the application domain. As with the system verification model, the fundamental aspects of such models include an evidence structure and a sampling approach for acquiring the evidence. Validation evidence comes primarily from testing for the accomplishment of the intended effects in the application domain, based on the stakeholder requirements model.

From among all the intended effects of the system in the application environment, appropriate operations of the environment and the system should be choreographed to determine that selected effects are actually generated by the system. Sampling of the system environment situations and use patterns in the application domain effects provide the cases to be examined.

The primary questions to be answered by this model pertain to adequacy in observation of intended effects and judgment of stakeholder requirement satisfaction