

Withdrawn Standard

IEEE Std 1175.3™-2004

IEEE Standards

1175.3™

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

IEEE Computer Society

Sponsored by the
Software and Systems Engineering Standards Committee



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

8 September 2004

Print: SH95222

PDF: SH95222



Documents

Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on February 24, 2026 at 11:14:19 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1175.3-2004

Withdrawn Standard

Recognized as an
American National Standard (ANSI)

IEEE Std 1175.3™-2004

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

Sponsor

Software and Systems Engineering Standards Committee
of the
IEEE Computer Society

Approved 8 July 2004

American National Standards Institute

Approved 25 March 2004

IEEE-SA Standards Board

Abstract: The purpose of this standard is to specify a common set of modeling concepts based on those found in commercial CASE tools for describing the operational behavior of a software product. This standard establishes a uniform, integrated model of software concepts related to software functionality. It also provides a textual syntax for expressing the common properties (attributes and relationships) of those concepts as they have been used to model software behavior. The 1175.3 exchange syntax is both machine-processable and human-readable.

Keywords: Computer-Aided Software Engineering (CASE) tools, Semantic Transfer Language (STL), software behavior concept, transfer syntax

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

Copyright © 2004 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 8 September 2004. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Incorporated.

Print: ISBN 0-7381-4001-5 SH95222
PDF: ISBN 0-7381-4002-3 SS95222

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

Withdrawn Standard

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied “**AS IS**.”

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331 USA

NOTE—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 shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Withdrawn Standard

Introduction

(This introduction is not part of IEEE Std 1175.3-2004, IEEE Standard for CASE Tool Interconnections—Reference Model for Specifying Software Behavior.)

The 1175 family of standards

This standard is a member of the 1175 family of IEEE standards. The members of this family are listed below.

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 balloted.

Standard number	Title
IEEE Std 1175.1™-2002	IEEE Guide for CASE Tool Interconnections—Classification and Description
IEEE P1175.2	Draft Recommended Practice for CASE Tool Interconnections—Characterization of Interconnections
IEEE Std 1175.3-2004	IEEE Standard for CASE Tool Interconnections—Reference Model for Specifying Software Behavior
IEEE P1175.4	Draft Standard for CASE Tool Interconnections—Reference Model for Specifying System Behavior
IEEE P1175.5	Draft Standard for CASE Tool Interconnections—Syntax for Transferring Behavior Specifications
NOTE—The products listed in this table are trademarks of the Institute of Electrical and Electronics Engineers, Incorporated.	

This family of standards replaces IEEE Std 1175™-1991, IEEE Standard Reference Model for Computing System Tool Interconnections.¹ 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 revision 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.

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 most computing system tools do not need to communicate behavior descriptions of subject systems, their creators need to develop such descriptions for them.

A brief summary overview of each of the members of this family of standards is given below. A more complete overview is available in IEEE Std 1175.1™-2002, 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.

¹Although approved in 1991, IEEE Std 1175-1991 was actually published in 1992, and it is sometimes found referenced as IEEE Std 1175-1992. It appears in the standards listing on the IEEE Xplore website (<http://ieeexplore.ieee.org/>) as "IEEE Std 1175, 17 Aug. 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 website.

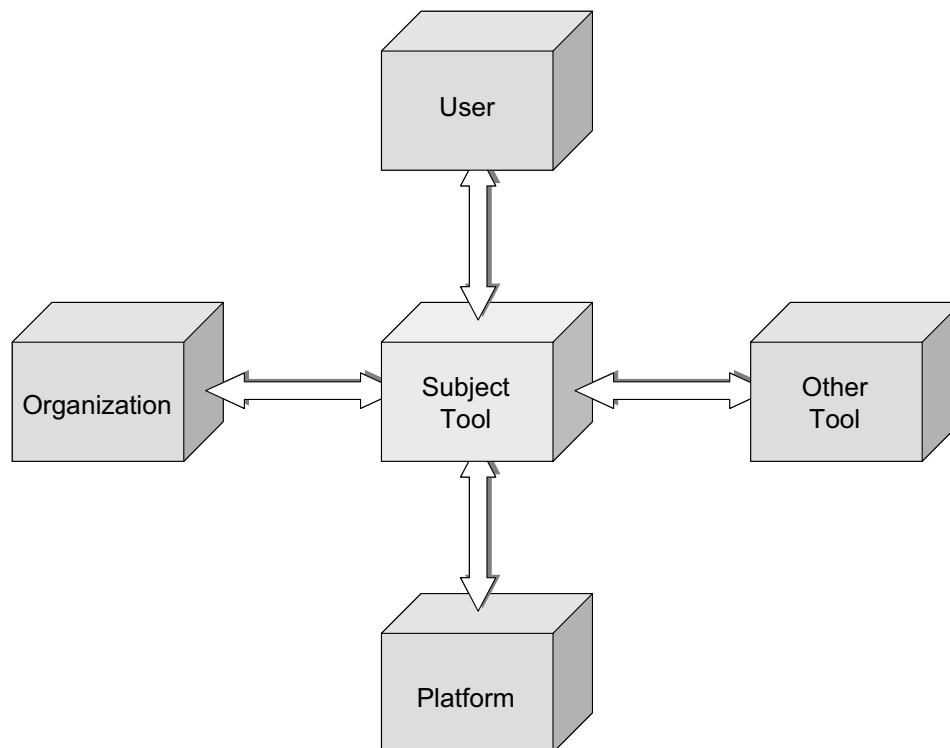
Withdrawn Standard

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

IEEE Std 1175.1-2002 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 that standard, and the interrelationships between the family members.

IEEE P1175.2 Recommended Practice for CASE Tool Interconnections—Characterization of Interconnections

IEEE P1175.2 presents four perspectives on a computing system tool's interconnections that offer insight into the operational problems of interconnecting computing system tools with their environment. IEEE P1175.2 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 the figure below. These contextual attributes are of the “news-story” form: who, what, when, where, 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, IEEE P1175.2 contributes to the effective implementation and application of computing system tools.



Withdrawn Standard

IEEE Std 1175.3-2004 Standard for CASE Tool Interconnections—Reference Model for Specifying Software Behavior (this standard)

This standard 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. This standard 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.

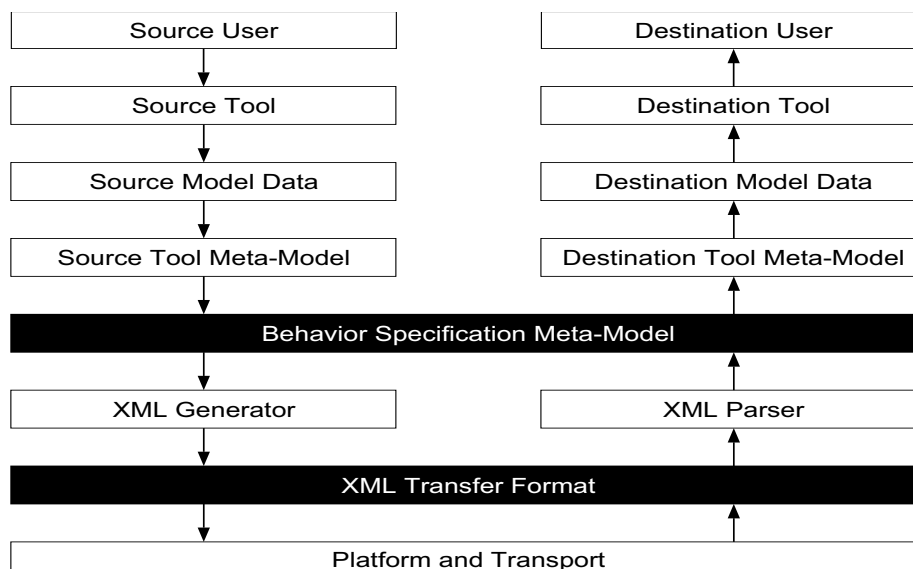
To permit backward compatibility with Part 3 of IEEE Std 1175-1991, this standard makes no changes to the STL syntax or to the rules for conformance to this syntax as originally defined in Part 3 of IEEE Std 1175-1991. However, some aspects of the 1991 syntax that were previously left as user defined 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, more “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.

IEEE P1175.4 Standard for CASE Tool Interconnections—Reference Model for Specifying System Behavior

IEEE P1175.4 encompasses the description of other computing systems supported by IEEE 1175.3-2004, but it goes further, providing a basis for representing a wider variety of computing systems. Specifically, IEEE P1175.4 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 Standard for CASE Tool Interconnections—Syntax for Transferring Behavior Specifications

IEEE P1175.5 provides a structured syntax and a specific set of data elements that can be exchanged as a text stream or text file with users or other tools. This syntax permits describing a system’s operational behavior as an instance of the reference meta-model in IEEE P1175.4. The figure below illustrates the relationship between the IEEE P1175.4 meta-model and the IEEE P1175.5 XML syntax in the context of the tool-tool exchange of a computing system description. IEEE P1175.5 also provides a logically equivalent human-readable syntax.



Withdrawn Standard

Organization of this standard

This document contains 11 Clauses and 6 Annexes. The annotations below provide a roadmap to the contents.

1	<i>Overview</i> —This clause describes the purpose and scope of this standard.
2	<i>References</i> —This clause lists the references necessary for implementing this standard.
3	<i>Definitions</i> —This clause provides definitions of terms.
4	<i>Introduction to Software Behavior Representation Concepts</i> —This clause provides an introductory overview of software behavior concepts as used within this standard.
5	<i>Conventions for Software Behavior Concept Specifications</i> —This clause describes a structured representation format for software concepts.
6	<i>Specification of Software Behavior Representation Concepts</i> —This clause provides a reference for software concepts. Each subclause in this clause provides a textual description of a software concept, followed by one or more tables specifying the concept definition template for the concept.
7	<i>Specification of Transfer Syntax for Software Behavior Representation Concepts</i> —This clause provides a syntax for representing, and transferring, in a computer-processable manner, a descriptive model of a software component that is based on the software concepts presented in Clause 6.
8	<i>STL Transfer File</i> —This clause defines the contents and structure of an STL Transfer File, constructed using the STL, containing a software behavior description.
9	<i>Tool Interconnection Profile</i> —This clause defines the format of, and provides instructions for preparing, a Tool Interconnection Profile. This Profile may be used to document the degree to which a tool's incoming and outgoing STL interchange files can support the full scope of coverage of the STL.
10	<i>Transfer Language Extensibility</i> —This clause describes how to define and document additional transfer language clauses, comparable with and compatible with the STL, to provide for the exchange of user-defined application information that is not addressed within the breadth and depth of coverage of the standard STL.
11	<i>Conformance</i> —This clause defines conformance to the Semantic Transfer Language.
Annex A	<i>(informative) Entity-Relationship Diagrams of the STL</i> —This annex provides diagrammatic illustrations of the major STL concepts and their interrelationships.
Annex B	<i>(informative) STL Entity-Relationship Diagram Example</i> —This annex provides an illustrative example of how the STL may be used to represent the contents of an entity-relationship diagram.
Annex C	<i>(informative) STL Data Flow Diagram Example</i> —This annex provides an illustrative example of how the STL may be used to represent the contents of a data flow diagram.
Annex D	<i>(informative) STL ESML and StateTransition Diagram Example</i> —This annex provides an illustrative example of how the STL may be used to represent the contents of a state transition diagram.
Annex E	<i>(normative) STL Clause Listing</i> —This annex provides a complete but succinct summary listing of each different kind of STL clause that may be used within an STL sentence. ^a
Annex F	<i>(informative) Bibliography</i> —This annex provides references for other standards whose potential usage is identified within this standard.

^aAn exception is the special S_Packet Identification Sentence, defined in Clause 8, which is not included in Annex E.

Withdrawn Standard

Notice to users

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 shall not be responsible for identifying patents or patent applications for which a license may be required to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Participants

The following is a list of participants in the 1175.3 Working Group.

	Carl A. Singer, <i>Chair</i>	
Jimi Arvidsson	Subramanya R. Jois	Horace H. Lawrence
Abby Beifeld	Paul C. Jorgensen	Lou Pinto
Bakul Banerjee	Sohel Majid Khan	Robert M. Poston
Peter L. Eirich	Dwayne L. Knirk	Robert Wessely
Álvaro Francisco de Castro	Ashraf Kottilungal	
Medieros		

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

Bakul Banerjee	Sourav Dutta	Lou Pinto
Edward Bartlett	Clint Early	Robert J. Schaaf
Richard Biehl	William Eventoff	Carl A. Singer
Juris Borzovs	John Fendrich	Luca Spotorno
Joseph Butchko	Yaacov Fenster	Scott Valcourt
Lawrence Catchpole	Juan Garbajosa	Paul Work
Antonio M. Cicu	William Junk	Oren Yuen
Guru Dutt Dhingra	Thomas M. Kurihara	Li Zhang
	Dwayne L. Knirk	

Withdrawn Standard

When the IEEE-SA Standards Board approved this standard on 25 March 2004, it had the following membership:

Don Wright, Chair
Steve M. Mills, Vice Chair
Judith Gorman, Secretary

Chuck Adams
H. Stephen Berger
Mark D. Bowman
Joseph A. Bruder
Bob Davis
Roberto de Boisson
Julian Forster*
Arnold M. Greenspan

Mark S. Halpin
Raymond Hapeman
Richard J. Holleman
Richard H. Hulett
Lowell G. Johnson
Joseph L. Koepfinger*
Hermann Koch
Thomas J. McGean

Daleep C. Mohla
Paul Nikolich
T. W. Olsen
Ronald C. Petersen
Gary S. Robinson
Frank Stone
Malcolm V. Thaden
Doug Topping
Joe D. Watson

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Alan Cookson, *NIST Representative*

Michelle Turner
IEEE Standards Project Editor

Withdrawn Standard

Contents

1.	Overview.....	1
1.1	Scope.....	1
1.2	Purpose.....	2
1.3	Relationships to object modeling standards.....	2
2.	References.....	3
3.	Definitions	3
4.	Introduction to software behavior representation concepts	5
4.1	Concept organization	5
4.2	Data concepts	8
4.3	Event concepts	10
4.4	Logic concepts	11
4.5	Transform concepts.....	11
4.6	Connection concepts	11
4.7	Evolution concepts.....	12
4.8	Quantifying: Value concepts	12
4.9	Scoping: Organization concepts	12
4.10	Mapping: Image concepts	13
5.	Conventions for software behavior concept specifications	13
5.1	Elements of software behavior concept specifications	13
5.2	Common attributes.....	15
5.3	Common relations.....	18
6.	Specification of software behavior representation concepts.....	19
6.1	Action.....	20
6.2	Collection	23
6.3	Condition	23
6.4	ConnectionPath	26
6.5	Constant	29
6.6	DataItem	33
6.7	DataKey	33
6.8	DataPart	37
6.9	DataRole	39
6.10	DataStore	39
6.11	DataType	44
6.12	DataView	52
6.13	EventItem	54
6.14	EventType	54
6.15	GraphicSymbol	56
6.16	Object	62
6.17	State	64
6.18	StateTransition	67

Withdrawn Standard

7.	Specification of transfer syntax for software behavior representation concepts	70
7.1	STL overview	70
7.2	STL goals	70
7.3	STL sentence form	71
7.4	STL description notation	72
7.5	STL language elements.....	73
7.6	STL sentences	80
7.7	STL language integrity	86
7.8	STL syntax summary	88
7.9	STL S_Packet summary.....	89
8.	STL transfer file.....	107
8.1	STL transfer file organization.....	107
8.2	STL information packet (S_Packet)	107
8.3	S_Packet identification sentence	107
8.4	STL transfer file illustration	108
9.	Tool Interconnection Profile	110
9.1	Overview of the Tool Interconnection Profile.....	110
9.2	Basis for the Tool Interconnection Profile.....	110
9.3	Tool Interconnection Profile sentence	111
9.4	Profile completeness and consistency.....	111
9.5	Profile example	111
10.	Transfer language extensibility.....	114
10.1	Overview of STL transfer file extensibility	114
10.2	Specification of extensibility	114
11.	Conformance.....	114
11.1	Conformance to standard STL	115
11.2	Conformance testing	115
	Annex A (informative) Entity-relationship diagrams of the STL.....	116
	Annex B (informative) STL entity-relationship diagram example.....	122
	Annex C (informative) STL data flow diagram example	129
	Annex D (informative) STL Extended Systems Modeling Language and state transition diagram example	136
	Annex E (normative) STL clause listing	142
	Annex F (informative) Bibliography	166

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

1. Overview

1.1 Scope

Computer-Aided Software Engineering (CASE) tools are used to describe the behavior of software using a variety of different design notations. These may be graphical or textual in nature, or they may be a combination. This standard provides a reference model of fundamental software concepts that form the “building blocks” for a number of these commonly used notations. This standard also includes a textual language, the Semantic Transfer Language (STL), for representing software application behavior descriptions. A software behavior description consists of a collection of sentences that conform to the formal syntax of the STL and that are to be interpreted in terms of the software concepts defined in this standard. The STL syntax is designed to be computer-parsable, while remaining easy for users to read and write. This reference model and transfer syntax may be used for directly recording, storing, and analyzing a software behavior description, as well as for transferring elements of a software behavior description between CASE tools.

This standard excludes the structural descriptions of software product designs or implementations, and it does not provide a traceable decomposition of software into programs, modules, subroutines, or objects. However, it *does* provide the means to describe the functions and behavior of a program, module, subroutine, or object.

This standard focuses on the semantics of information concerning software behavior, not on graphical representations for that information. Graphical information is not important when communicating among tools that do not need to reproduce the same drawing. However, the STL defined in this standard does include references to images or drawings of graphical symbols that may be used to indicate correspondences between graphical and textual representations of the same software concepts.

1.2 Purpose

The purpose of this standard is to specify a common set of modeling concepts based on those found in commercial CASE tools for describing the operational behavior of a software product. This standard establishes a uniform, integrated model of software concepts related to software functionality. It also provides a textual syntax for expressing the common properties (attributes and relationships) of those concepts as they have been used to model software behavior. The 1175.3 exchange syntax is both machine-processable and human-readable.

As an alternative to establishing direct tool–tool mappings as the basis for information sharing among CASE tools, this standard identifies a common set of “tool-neutral” modeling concepts. By mapping their internal data into the common STL syntax provided by this standard, and by exchanging data files constructed using the STL, it becomes possible for two different CASE tools to reliably exchange descriptions of a subject software application. As each tool can map its own meta-model constructs against the standard 1175.3 software behavior concepts, it is not necessary for any one tool to understand the internal structures of another tool. This also means that a number of tools can exchange descriptions on the basis of one set of concept mappings per tool, a substantial reduction of effort from the pairwise mapping approach.

In addition to tool–tool information exchange applications, the STL has also been found to be useful for design reviews, a form of tool-user interconnection. Because the STL is straightforward to write, and is much like everyday English sentences, it is also possible to employ the STL for user–user and user–tool information exchange applications.

This standard incorporates and extends the material in Part 3 of IEEE Std 1175™-1991 [B5].¹ For backward compatibility, this standard does not change any of the syntax or conformance aspects of the STL as defined in Part 3 of IEEE Std 1175-1991 [B5].² The primary emphasis for this revision has been to improve the clarity and usability of the material for its original intended purposes, rather than to revise the content or otherwise update it to incorporate and represent the software methodology developments of the 1990s.

1.3 Relationships to object modeling standards

The STL defined in this standard complements the Unified Modeling Language (UML) [B18], the Extensible Markup Language (XML) [B3], the specification for XML Metadata Interchange (XMI) format [B19], and the OMG Meta-Object Facility (MOF) [B14]. Those standards and specifications together define how UML models and other MOF-compliant models can be passed from tool to tool, using the format specified by XMI. Those standards and specifications do not apply for models defined using methodologies that are not MOF-compliant. However, the concepts defined in this standard do encompass the meaning of the elements found in a variety of common modeling techniques, and those elements can be mapped to the STL concepts to create a basis for tool–tool exchange of models. In this manner, the STL can be used to transfer models that are outside the scope of XMI.

Although this standard provides a basic capability to represent objects, it does not encompass many of the specific features incorporated into UML and does not replace XMI for its intended uses. As the scopes of application for STL and for XMI have only limited overlap, these two formats are not substitutes for one another. Rather, each provides value to its own intended domain of usage.

¹The numbers in brackets correspond to those of the bibliography in Annex F.

²However, additional syntax has been added to provide a standardized form for conditional expressions. These were left as a user-defined item in IEEE Std 1175-1991 [B5].

2. References

This standard shall be used in conjunction with the following publications. When the following publications are superseded by an approved revision, the revision shall apply. The following glossary standard shall be used, when applicable, for software behavior concept terms not defined in this standard.

IEEE Std 610.12™-1990, IEEE Standard Glossary of Software Engineering Terminology.^{3, 4}

3. Definitions

For the purposes of this standard, the following terms and definitions apply.⁵ Additional relevant terms defined in IEEE Std 610.12-1990⁶ include *data*, *interface*, *literal*, *semantics*, *software tool*, and *syntax*. The *Authoritative Dictionary of IEEE Standards Terms*, Seventh Edition [B4] should be referenced for terms not defined in this clause. The ISO/IEC international standard references on information technology vocabulary, ISO/IEC 2382-1:1993 [B10], may also be helpful.

3.1 Action: A process of transformation that operates upon data or other types of inputs to create data, produce outputs, or change the state or condition of the subject software. [STL]

3.2 Collection: An unrestricted grouping of software behavior concept instances into a particular named subset. [STL]

3.3 condExpression: The syntax used in the STL both for defining simple Conditions and for compounding simple Conditions to define more complex Conditions.

3.4 Condition: A true or false logical predicate. [STL]

3.5 ConnectionPath: A pathway for the propagation of information. [STL]

3.6 Constant: A reference name for a numerical, lexical, or externally specified fixed (unchanging) value. [STL]

3.7 DataItem: A possible value or structure of values that can be retained or processed. [STL]

3.8 DataKey: An identifier for a particular grouping of data. [STL]

3.9 DataPart: A component of a structured DataType with a domain specified by reference to some other DataType. [STL]

3.10 DataRole: A characterization of the way an entity DataType participates in a relationship. [STL]

3.11 DataStore: A specification of data retention capabilities for the subject software. [STL]

3.12 DataType: A set of possible values or structures of data. [STL]

³IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (<http://standards.ieee.org/>).

⁴This reference is required only for definitions of the terms: *data*, *interface*, *literal*, *semantics*, *software tool*, and *syntax*. It may be referenced for other terms as needed.

⁵The majority of these terms indicate STL concepts (see 3.27). These terms are designated by having “[STL]” at the end of the definition text, and they are further explained and specified in Clause 6.

⁶Information on references can be found in Clause 2.

3.13 DataView: A partitioning of a supertype DataType entity into subtype entity. [STL]

3.14 Event: A change detectable by the subject software.

3.15 EventItem: An occurrence of an EventType. [STL]

3.16 EventType: A set of possible times or time periods at which an Event may occur. [STL]

3.17 GraphicSymbol: A reference to a particular image or drawing for representing a particular software behavior concept instance. [STL]

3.18 logic: Logical restrictions (Conditions) on Actions.

3.19 Object: A structure of encapsulated behaviors with visible interactions and messages. [STL]

3.20 S_Packet: A logically coherent grouping of STL Sentences that describes a set of software concepts.

3.21 Semantic Transfer Language (STL): The language defined in this standard for the purpose of representing software application behavior descriptions.

NOTE—The Semantic Transfer Language has a formal syntax that is computer-parsable, while remaining easy for users to read and write.⁷

3.22 Semantic Transfer Language (STL) concept: A software behavior concept that is represented in the STL.

3.23 Semantic Transfer Language (STL) clause: A portion of an STL sentence that describes and characterizes an attribute of, or a relationship for, a software behavior concept.

3.24 Semantic Transfer Language (STL) sentence: A statement conforming to the syntax of the Semantic Transfer Language (STL) that describes and characterizes a software behavior concept recognized within the STL.

3.25 software behavior concepts: Types of quantifiable properties and relationships whose instances may be used to describe the dynamic behavior of software execution.

3.26 State: An identified condition or set of conditions within the subject software that is associated with certain of the possible actions of the subject software. [STL]

3.27 StateTransition: A change from one State to another. [STL]

3.28 subject software: A computing (software) application (existing or to be created) about which descriptive information is being developed in a computing system tool or a Computer-Aided Software Engineering (CASE) tool.

⁷Notes in text, tables, and figures are given for information only, and do not contain requirements needed to implement the standard.

4. Introduction to software behavior representation concepts

This standard defines an STL for describing software behavior. Clause 4, Clause 5, and Clause 6 together provide a description of the semantics for the language and provide syntactic templates for the sentences of the language. This clause and Clause 5 form an introductory overview of the concepts represented in STL, called STL concepts. Clause 6 provides software behavior concept descriptions in the form of a reference handbook, with each STL concept presented in alphabetical order.

4.1 Concept organization

The STL is constructed using a set of fundamental software concepts. These STL concepts are logically organized according to subject, purpose, and type. This logical organization gives a classification hierarchy, with three levels, that shows both the commonality that exists among the concepts and the differences that distinguish among them. This classification is exhaustive, and it provides the distinctions that make the STL concepts mutually exclusive as well. Thus, each concept fills a well-defined information transfer need.

This classification is described in the remainder of this clause. Each STL concept instance has a subject about which it provides information. Two categories of STL concepts are distinguished as follows:

- The software behavior category
- The interpretive support category

The software behavior category contains the concepts whose instances provide facts about the behavior of the subject software being represented with an STL description. These are the core concepts of the STL. They are used to express the specific observable characteristics of the software in its application. As illustrated in Figure 1, various models may be abstracted (the Viewpoint level) and various diagrams may be made (the Presentation level) from the instances of these concepts that may be found in an STL software description (the Concept level).

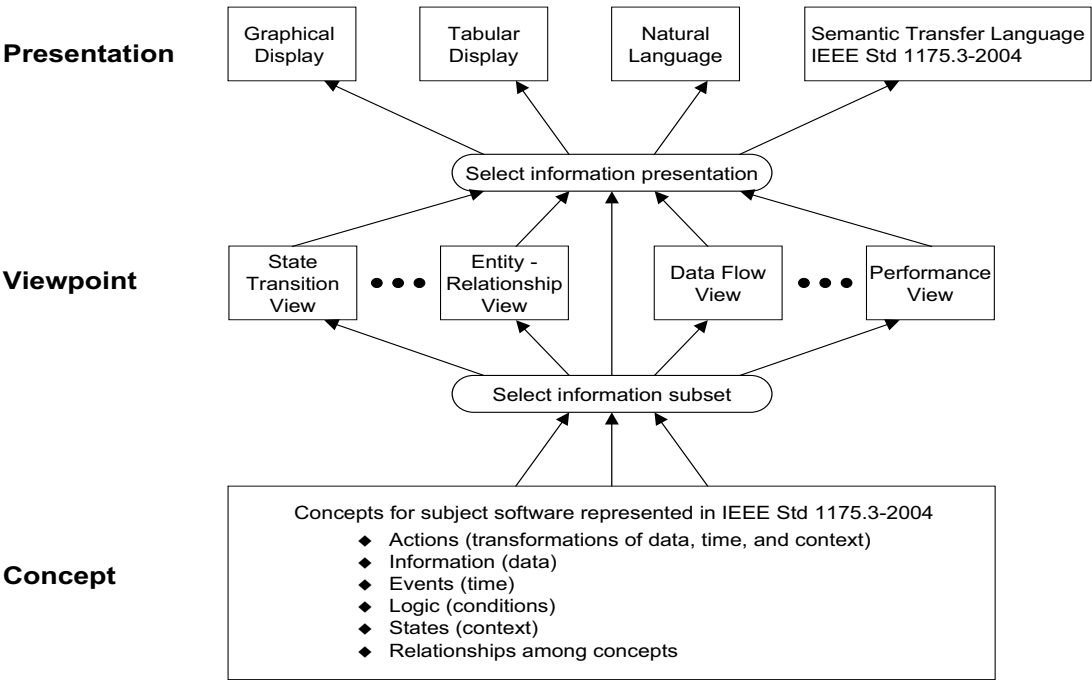


Figure 1—Presentations, viewpoints, and concepts

The interpretive support category contains concepts whose instances provide facts about the quantification, organization, and presentation of other concept instances in an STL software description. Instances of concepts in the interpretive support category provide no direct facts about the software behavior being described. They facilitate the user interpretation and application of the information provided by the software behavior concept instances.

The organization of concepts within these two categories is described below in 4.1.1 and 4.1.2.

4.1.1 Software behavior category

STL concepts in the software behavior category express the ideas or thoughts that a product developer may want to record concerning the operation of a software product or a software component within a system. Software behavior category concepts address the operations of software and correspond to the Concept level in Figure 1.

Six groups of STL behavior description concepts are distinguished on the basis of the following purposes:

- a) *Representing application properties.* Software manipulates representations of application properties. These representations are called “data.” The specification of software behavior must therefore refer to such data representations. The STL provides six data concepts whose instances are used to describe those representations. They are the DataItem, the DataType, the DataPart, the DataKey, the DataRole, and the DataView concepts.
- b) *Representing times.* Some software behavior requires explicit responses (synchronization) to things that occur external to it. These things are occurrences that affect the operation of the software but that are not dictated by the software's own flow of control. They are called “Events.” A description of software response to Events needs the notion of time points and intervals. The STL provides two Event concepts whose instances are used to describe the occurrences of such time points. They are the EventType and the EventItem concepts.
- c) *Expressing decisions and assertions governing transforms.* The ability of software to deal differently with different situations is a result of its ability to compare things and make decisions. The same kind of logic with which those decisions are expressed is also used to express assertions on the results of the software behavior. The STL provides the Condition as a basic logic concept. The STL provides a syntactic element, the condExpression, as a means of expressing the logical definition of a condition (including provision for Boolean compounding of conditions or compounding of algebraic relations on values). Both condExpressions and instances of Conditions are used to describe decisions and assertions. (There is no STL concept definition sentence for condExpression, which is incorporated into the STL as a syntactic element.)
- d) *Transforming property representations.* The primary accomplishments of software operations are to calculate upon and to transform the data representations of application properties. Software operations may use inputs occurring at one time to produce outputs occurring at another time, effectively accomplishing a change (or transformation) of inputs into outputs. In the important special case of data storage, operations may result in no change in the stored data from one time to another time (an identity transform). The STL provides two transform concepts whose instances are used to describe such effects of software operation. These are the Action concept and the DataStore concept.
- e) *Propagating transform effects.* For the operation of software to have any useful effect in the external world, it must be connected to a source for input and a destination for output. Likewise, for the effects of an operation at one time to be referenced by an operation at a later time, both must be connected to some store into which the effect of the first can be deposited and later retrieved. This description of software behavior in terms of transformation and propagation is common in current practice. The STL therefore provides a concept whose instances are used to describe the propagation of information from source to destination. This is the ConnectionPath concept.

- f) *Sequencing transforms*. Associated with the monotonic increase of world time is the notion of a coordinated evolution of the software and its external world through a succession of operations. The description of this evolution may involve both sequences of transforms and sequences of property representations. The STL provides a pair of evolution concepts. Their instances are used to describe individual sets of transforms and property representations, and to describe the allowed sequences in which those sets can occur. They are the State concept and the StateTransition concept.

Table 1 summarizes the Software Behavior Category concepts. The subject of these concepts is the operational aspects of software.

Table 1—Classification hierarchy for Software Behavior Category concepts

Category group (content)	Purpose (type of information)
Data group	Purpose: property representation
DataItem	labeled appearance of a DataType
DataKey	property representation identifier
DataType	property representation abstraction
DataPart	property component structuring
DataRole	rules for a relationship instance
DataView	entity subtype structure
Time group	Purpose: time representation
EventItem	labeled appearance of an EventType
EventType	time representation abstraction
Logic group	Purpose: decision representation
Condition	named property value instance discriminators
Transform group	Purpose: changes of property representation
Action	observable transformations of properties
DataStore	retention of properties between Actions
Coupling group	Purpose: propagation of changes
ConnectionPath	information propagator between transforms ^a
Evolution group	Purpose: sequencing of changes
State ^b	property characterization or transform access
StateTransition	allowed changes of State

^aActions or Datastores.

^bDataState or ActionState.

4.1.2 Interpretive support category

The interpretive support category is divided into three groups that serve to quantify, scope, or map the software behavior concepts.

- a) *Quantifying concept instances.* All concept instances are quantified by value concept instances. The STL provides the Constant as the basic value concept. (The STL also provides a syntactic element, the literal, that is used to define an un-named value.) Several value subtypes are provided in the STL that, in conjunction with the STL Data concepts, are used to quantify application-specific facts.
- b) *Scoping intellectual viewpoints on the concept instances.* Much information is needed to specify adequately the required behavior of modern software. Many different approaches have been developed to focus on selected parts of the information. These approaches normally adopt a particular viewpoint for selecting the information to consider. They also include hierarchical partitioning of the selected information into intellectually manageable chunks that are easier to reason about. The STL provides two organization concepts that are used to express these subsets and their hierarchical structures. They are the Collection concept and the Object concept.
- c) *Mapping concept instances to diagram images.* Many current software design approaches also have diagrammatic representations for the information concerning the software. As illustrated in Figure 1, ISO 5807:1985 [B7] and ISO 8790:1987 [B9] define graphic symbols commonly used in software engineering diagrams. The STL does not include transfer of graphical information directly. However, the STL does have a concept for mapping between software behavior concept instances and externally defined images or drawings. That concept is the GraphicSymbol.

Table 2 illustrates the classification hierarchy for the Interpretive Support Category concepts, which address quantifying, scoping, and picturing the elements of software descriptions.

Table 2—Classification hierarchy for Interpretive Support Category concepts

Category group (concept)	Purpose (type of information)
Value group	Purpose: value instance representations
Constant	reference name for a value
Organization group	Purpose: user focus
Collection	grouping of software description elements
Object	encapsulation of properties and behaviors
Image group	Purpose: visual presentations in drawings
GraphicSymbol	reference to a named symbol instance

4.2 Data concepts

The STL has six concepts for data: the DataItem, the DataType, the DataPart, the DataKey, the DataRole, and the DataView.

A DataItem concept instance describes a single occurrence of data at one place and time in the subject application. A DataItem labels the occurrence so it can be referenced in connection, evolution, logic, or transform concept definitions.

The actual data occurring as a *DataItem* may be only one value at one point in time. Over time the *DataItem* may be any of several different values. The set of all possible values is called the domain of the *DataItem*. Each data occurrence is a member of that set.

A *DataType* concept instance is used to describe such a set of data values in the subject application. A *DataItem* is called an instance of a particular *DataType* when that *DataType* specifies the set of all possible data values that can occur for that *DataItem*. The *DataType* specifies the domain of the *DataItem*.

The members of a domain may be primitive data values or structured data values. An instance of an unstructured domain is an unstructured primitive value, such as an integer or a character. An instance of a structured domain is one or more component values, each from a component domain.

A structured *DataType* domain is described with a structure of *DataParts*. Each component of a structured *DataType* is defined as an instance of the *DataPart* concept. The name of a *DataPart* is the name of a component in the structure. The possible values for the component are specified by reference to some other appropriate lower level *DataType*. The structuring of *DataTypes* is thus hierarchical.

Within one structured *DataType* definition, the components are given unique *DataPart* names. However, any number of *DataParts* may refer to the same lower level *DataType* for the specification of the components' values.

Formats and units can be specified several ways. Where they are specified for a *DataType*, they are assumed to apply to any *DataItem* that is an instance of that *DataType* and to any *DataPart* that is an appearance of that *DataType*. Where they are specified for a *DataItem* or a *DataPart*, they are assumed to supersede any specification of the referenced *DataType*.

DataKey concept instances specify the identifiers for an entity or for a table *DataType*. A *DataKey* is a reference to one or more *DataParts* in a structured *DataType* whose data instance(s) serve uniquely to label value instance(s) of the entity definition.

DataRole concept instances provide the rules for relationships (interactions) among entity *DataTypes*. Relationship *DataTypes* involve *DataRoles*. The *DataRole* identifies an entity *DataType* participating in the relationship and defines its rules of participation.

DataView concept instances specify subtype decompositions for an entity *DataType*. One entity *DataType* may have several subtype decompositions. Each *DataView* is regarded as an alternative way of partitioning entity instances.

The following are some examples of *DataItems*, *DataTypes*, and *DataParts*:

- An input dialog prompts for “last name.” For one person, the data entry operator enters “Cawtajupac,” for another person, “Kuvercin.” These are particular values of *LastNameEntry*. *DataItem* *LastNameEntry* stands for any particular last name entry the operator makes. A large range of character strings can be entered. This is documented by defining the application-specific *DataType* *NameString* as an array of varying size from 1 to 20, say, of *NameChar*. Any occurrence of the *DataItem* *Last-NameEntry* is an instance of the *DataType* *NameString*.
- An input dialog prompts for “year of birth.” For one person, the data entry operator enters “1945,” for another person, “1957.” These are particular values of *BirthYearEntry*. *DataItem* *BirthYearEntry* stands for any particular year of birth entry the operator makes. Only a small range of years in the current epoch can be entered. This is documented by defining the application-specific *DataType* *Year* including values from 1900 to 2100 by 1. Any occurrence of the *DataItem* *BirthYearEntry* is an instance of the *DataType* *Year*.

- The actual scenario for data entry requires a last name, a first name, and an age. DataItem PersonEntry stands for any particular triplet entry. This is documented by defining a DataType Person as a sequence of three DataParts: LastName, FirstName, and BirthYear. A PersonEntry is an instance of DataType Person. The possible values for DataPart LastName are specified by DataType NameString. The possible values for DataPart FirstName are also specified by DataType NameString. Values of BirthYear are specified by DataType Year.

The difference between DataType, DataItem, and value is illustrated in Table 3.

Table 3—Illustration of DataType, DataItem, and data value instance

A DataType defines the set of possible values	A DataItem names an appearance of a DataType, and could have any one of the possible values in the set	A user entry supplies only one particular value—a value instance
NameString	LastNameEntry	Kuvercin
NameString	FirstNameEntry	Joe
Year	BirthYearEntry	1950
Person	PersonEntry	Kuvercin, Joe 1950

The data instance is an application-meaningful arrangement of one or more data values. It is a specific value for a DataItem. It can be any one of the values represented by the DataItem. The possible DataItem values are specified in the DataType definition. In particular, the instance of DataType Person in the example is an instance of the sequence structure, which is a sequence of one instance of each DataPart.

4.3 Event concepts

The STL has two concepts for Events: the EventItem and the EventType.

An EventItem concept instance describes a single occurrence of an Event at one place and time in the subject application. It labels the occurrence so it can be referenced in connection and transform concept definitions.

The actual Event occurring as an EventItem is some kind of detectable change that may occur at different times. The set of times at which the Event may occur is called the domain of the EventItem. Each time of occurrence is a member of that set.

An EventType concept instance provides a way to describe the time intervals for a kind of Event. This includes time points, which are the smallest intervals of recognition in the application. An EventType concept instance specifies a set of possible time intervals (points) for an Event that may be noticed at one or more places in the software operation. “Noticed” means that one Action causes some change to which another Action can synchronize and respond. The duration of such a change is an interval of time that is usually much shorter than the times associated with the subject Actions; it is a time point.

EventType instance definitions identify the Events that mark time for Actions to synchronize.

The difference between EventType, EventItem, and Event is illustrated in Table 4.

Table 4—Illustration of EventType, EventItem, and Event occurrence

An EventType defines the set of possible times at which a certain kind of Event can occur	An EventItem names an appearance of an EventType, and it could occur at any one of the possible times for an occurrence of that kind of Event	When an Action occurs, it does so at one particular time—an Event occurrence
all possible “UP” button pushes	any one push of the “UP” button	the particular “UP” button push at time 07:00:22
NOTE—The time instant 07:00:22 could have been any application-meaningful time point. It is a specific occurrence of an EventItem.		

4.4 Logic concepts

The STL has two representations for logic: the condExpression and the Condition.

For the condExpression, there is no STL concept definition sentence. A condExpression is an inline occurrence of a Boolean logic expression. A condExpression is recognized from standardized syntactic forms of expression (see 7.5.6). The structuring of condExpressions is hierarchical.

The STL provides a concept definition sentence for defining instances of the Condition concept. The principal distinction between a Condition and a condExpression is that a Condition has a symbolic name by which its defining expression can be referenced.

Both condExpression and Condition instances describe comparisons among one or more representations of application properties.

4.5 Transform concepts

The STL has two concepts for transform: the Action and the DataStore.

The Action concept provides a way to specify transformations and computations on data items. An Action concept instance is defined by its inputs and outputs and its context for occurrence.

The DataStore concept provides a way to specify retention of data items between Actions. A DataStore concept instance is defined by its inputs and outputs and by what it holds.

Actions and DataStores are alike in that they both have inputs and outputs and represent software behavior over time. They are different in that Actions transform data but DataStores may not transform data.

4.6 Connection concepts

The STL has one concept for connection: the ConnectionPath.

ConnectionPath concept instances describe paths for the propagation of information to transforms.

4.7 Evolution concepts

The STL has two concepts for evolution: the State and the StateTransition.

The State concept provides a way to represent contexts of the subject software that can be established by one or more sequences of Actions. Each State concept instance describes a different context in which Actions may occur.

The StateTransition concept provides a way to specify the allowed sequences of States. Every StateTransition is caused by some Action of the subject software. Each StateTransition concept instance identifies one allowed pairing of States.

4.8 Quantifying: Value concepts

The STL quantifies value either with anonymous values (literals) or with named, referenceable values (Constants). Constants are defined in terms of literals.

Values in the STL have the following subtypes:

- integer
- real
- character
- string
- stringshort
- stringlong
- structured
- external

All subtypes except the last can appear in the STL as either literals or Constants. The last value subtype, externalValue, can appear only as a Constant. See 7.5.5 for the definitions of literals. Attributes in STL concept instances are quantified by supplying either specified keywords or instances of the value subtypes, as appropriate.

4.9 Scoping: Organization concepts

The STL has two concepts for organization: the Collection and the Object.

Collection concept instances are used to group software behavior concept instances into named sets. In addition, Collections can be arranged into hierarchical structures. Collections correspond to the Viewpoint level illustrated in Figure 1. A viewpoint is a representation that reflects a system from a particular point of view. A named Collection may represent a complete or partial model of the software behavior.

The Object concept expresses the aggregation and encapsulation of application properties and transforms. Object instances are defined in terms of software behavior concept instances. Objects differ from Collections in that Objects may be related through inheritance and specialization, whereas Collections may not.

4.10 Mapping: Image concepts

The STL has one concept for mapping an instance of a software behavior concept to an externally defined image instance: the GraphicSymbol.

A GraphicSymbol definition does not define the graphical rendition of a symbol used to depict a concept instance. Rather, it links an STL concept instance name, attached to a concept instance within a view, to an external predefined symbol.

5. Conventions for software behavior concept specifications

Each STL concept is described in a separate subclause within Clause 6. This Clause explains the format by which the STL concepts are presented in Clause 6. Please refer to Table 1 and Table 2 for a listing of the STL concepts.

5.1 Elements of software behavior concept specifications

Each STL concept is described in Clause 6 with a separate subclause. The subclause begins with a short (in-line) summary table with the concept definition and other characterizations. This is followed by text that discusses the nature and purpose of each STL concept. Next, there is one table with the first three parts illustrated in Table 5. In addition, there may be a second table with one or more Subtype Text Presentation entries, also as illustrated in Table 5.

Table 5—Concept definition template

Concept name	Each concept is given one name
Concept meaning	A brief paraphrase of the definition and other summary explanations is given
Concept text presentation	Each concept is given a text presentation. The text presentation is divided into four parts: sentence keyword, sentence identifier, possible attributes, and possible relations. A “notes” part may be used to provide additional information
Subtype text presentation	Each subtype of the concept is given its own text presentation. The text presentation is divided into two parts: possible attributes and possible relations. A “notes” part may be used to provide additional information

Every STL concept definition must have a sentence keyword and a sentence identifier. The sentence keyword identifies the software concept that is the subject of an STL sentence (see 7.6.1), and a sentence identifier provides a symbolic term used as the unique identifier of the sentence in order to permit unambiguous processing of STL sentences (see 7.6.2).

An STL concept definition may also have possible relations and possible attributes listed under those headings. These indicate the relations (relationships) and attributes that are recognized in the STL as being potentially applicable for an instance of the software concept. Below the possible attributes line are the attribute keywords applicable to the concept. Below the possible relations line are the relation keywords applicable to the concept.

Where concepts have subtypes, the possible relations and possible attributes are partitioned into (a) the common ones that are allowed for all subtypes of the concept and (b) the unique ones that are allowed for one or more possible subtypes, but not for all of them. These are presented in subtype text presentation sections within each concept definition.

If there are no possible relations or possible attributes for a concept or a subtype, then “none” appears in the concept definition template in place of the attribute keywords or relation keywords, respectively.

The application of the concept sentence templates is illustrated in Table 6 with a simple example.⁸

Table 6—STL concept template example

concept name:	Action
concept meaning:	a transform, internal or external to the subject software, using and producing DataItems and causing StateTransitions
concept text presentation:	
sentence keyword:	Action
sentence identifier:	ActionID
possible attributes:	
has label	label_string
is actiontype	internal external
possible relations:	
uses DataItem	DataItemID { , DataItemID }
notes:	
(1) The label_string is a stringValueShort.	
subtype text presentation:	<i>for actiontype internal (InternalAction)</i>
possible attributes:	
is actiontype	internal
has process method	method_description
possible relations:	
is allowed in State	StateID { , StateID }
subtype text presentation:	<i>for actiontype external (ExternalAction)</i>
possible attributes:	
is actiontype	external
possible relations:	
none	

There are some relations and attributes that occur frequently for multiple STL concepts. These frequently appearing relations and attributes are discussed individually in 5.2 and 5.3.

In Table 6 and in the concept descriptions in Clause 6, the vertical bar symbol “|” indicates an either-or choice—one and only one of the entries separated by vertical bars in a possible attribute must be selected when stating an instance of that attribute in an STL sentence. Entries contained within a pair of “{” and “}” brackets may occur zero or more times in an instance of an STL sentence.

⁸This example has been formed from the Action sentence template in 6.1. It was created by eliminating all but one of the STL clauses for Action in each part of the template.

5.2 Common attributes

All attribute entries in the concept definition templates are of the form “has aPropertyName value_expression” or “is aType aTypeKeyword.” In some cases, the value expression is a single mnemonic; in others, it is an expression containing keywords and mnemonics. The underscore character “_” is included in all mnemonics names in the concept definition templates. This mnemonic is called an attribute keyphrase.

In an instance of a concept definition, each mnemonic is to be replaced by an appropriate type of literal value or defined Constant. The “notes” part of each concept definition specifies the allowed value type for the mnemonics used in the concept definition.

Each concept definition template specifies the attributes with which to specify instances of that concept. However, several kinds of attributes are common to several of the concept definitions: formats, units, labels, and descriptions. These are explained in the following subclauses.

5.2.1 Formats

Several data concept definitions contain a “has format” attribute. This attribute provides a string value describing the required display or printed appearance of values of the associated concept instance.

There is no STL interpretation of format specifications because such specifications are typically governed by tool or user needs. However, to transfer format information from one tool to another, some common specification form must be used. This standard recommends using the C programming language (ISO/IEC 9899:1999 [B11]) “sprintf” format control strings. Communicating tools may establish other standards for format specification by adopting such specifications from standards for other programming languages or from other tools.

5.2.2 Units

Several concept definitions contain some form of “has units” attribute. (In a units attribute, the last word in the attribute keywords is “units.”) A units attribute provides a string value describing a unit of measure according to which specified quantities are to be expressed. These include time units for measures of time and volume units for measures of content. Quantities having specified units of measure always have real values to accommodate possible scaling in the units.

There is no STL interpretation of unit of measure. Unit specifications are usually very application specific. However, to transfer numeric information from one tool to another, some common units of measure must be used. This standard recommends using the units defined in NISTIR 4412, The Initial Graphics Exchange Specification (IGES) Version 3.1.1, published in September 1991 [B17]. These are supplemented by two additional units for use in this standard: TimeStamp and Count.

Units of measure are categorized by type. The types are Distance, Mass, Duration, Electric Current, Thermodynamic Temperature, Amount of Substance, Luminous Intensity, Plane Angle, Solid Angle, TimeStamp, and Count. For each type except Count, there is a specified set of allowed units. The value string for a units attribute is constructed by concatenating the type and value abbreviations with a hyphen separator. For example, a length measured in units of foot is “DISTANCE-FT.” These types, their allowed units, and their designated abbreviations are listed in Table 7.

The units in Table 7 are taken from the IGES 5.1 standard [B17], except for the addition of TIMESTAMP and COUNT.

The TIMESTAMP type, not included in IGES 5.1 [B17], is defined in ISO 8601:2000 [B8].

As counts of things are so common in software specifications, this standard also provides a Count type. A complete unit specification is given by concatenating a name for the kind of thing being counted. For example, the unit for the number of packages in a shipment might be “COUNT-PACKAGES,” and the volume of a DataStore might be “COUNT-BYTES.”

Compound units are expressed in algebraic form. Expressions may contain constants and the operators “*” for multiplication, “/” for division, and “**” for exponentiation. Parentheses may be used to show precedence. Thus, for example, acceleration in kilometers per hour per hour is denoted as “(1/1000)*DISTANCE-M/(TIME-H**2).” Scale factors (e.g., 1000 for kilometers) may be included in the expression for units.

Table 7—Units of measure

Type of unit		Value of unit	
Category	Indicates	Abbreviation	Indicates
DISTANCE	Length; spatial separation	A	Angstrom
		AU	Astronomical unit
		FT	Foot
		IN	Inch
		LY	Light year
		M	Meter
		UM	Micron
		MIL	Mil (0.001 inch)
		MI	Mile
		KN	Nautical mile
		Y	Yard
MASS	Mass	C	Carat
		DR	Dram
		GA	Grain
		KG	Kilogram
		MT	Metric ton
		OU	Ounce
		LB	Pound
		S	Slug
CURRENT	Electric current	A	Ampere
TEMPERATURE	Thermodynamic temperature	C	Centigrade
		F	Fahrenheit
		K	Kelvin
		R	Rankine

Table 7—Units of measure *(continued)*

Type of unit		Value of unit	
Category	Indicates	Abbreviation	Indicates
DURATION	Time interval	D	Day
		HR	Hour
		M	Minute
		S	Second
		W	Week
		Y	Year
AMOUNT	Amount of substance	M	Mole
INTENSITY	Luminous intensity	C	Candela
PLANE	Plane angle	D	Degree
		G	Grad
		M	Minute
		R	Radian
		REV	Revolution
		S	Second
SOLID	Solid angle	C	Steradian
TIMESTAMP	An instant in time	T	Time of day
		D	Date
		S	Time stamp
COUNT	Item count	* ^a	Kind of item

^aCount units have no standard abbreviations; a meaningful name for the kind of item being counted should be supplied.

5.2.3 Labels

Every concept definition may contain the “has label” attribute. This attribute provides a short string value called the label string for a concept instance definition. This allows an STL writer to attach a label or name to the instance. Often that label is the one put on a graphics image in a diagram.

5.2.4 Descriptions

Every concept definition may contain the “has description” attribute. This attribute provides a long string value called the description string for a concept instance definition. This allows the STL writer to attach free-form text to the instance.

5.3 Common relations

Different concept definitions provide different relations with which to specify instances of those concepts. Relation entries in the concept definition templates for the following kinds of relations are common to several of the concept definitions:

- Aggregation
- Association
- Specialization
- Collection
- Presentation

These are explained in the following subclauses.

5.3.1 Aggregation relations

Many of the concept definitions allow the definition of structured instances. That is, instances of the concept may be defined in levels of structure or aggregation. These are “structural hierarchies.” For example, a “person” DataType may be an aggregate of several component DataParts such as name, address, and so on.

Structural relations are identified by the keyword “component” as in “Action A01 has component Action A02, A03.” The inverse statement is “Action A02 is a component of Action A01.”

5.3.2 Association relations

Most concept instances are associated with other concept instances in many different organization structures. For example, a ConnectionPath “carries” a DataItem, and an Action “uses” a DataItem. Associations like this are the threads that tie the individual concept instances into an integrated expression of the subject software's behavior.

The actual types of such relations are many and varied. These associations are identified by the relation keyphrases presented for each concept definition in the rest of this clause.

5.3.3 Specialization (inheritance) relations

For some STL concepts, instances are specified most effectively through classification hierarchies. These are groupings based on a degree of specialization. For example, an entity DataType called “vehicles” may have specializations called “car,” “truck,” and “trailer.” The vehicle DataType is the less specific, more general, and more abstract. The specializations are said to be the more specific, less general, and less abstract. The concept instances that are the more general are often called “supertypes.” The concept instances that are the more specific are often called “subtypes.”

When the definition of a concept instance at one level of abstraction specializes the definition of a concept instance at a higher level, the lower level concept instance is said to “inherit” the characteristics of the higher level concept.

In the STL, the more specific instances are identified in the more general instance definition by the “has subtype” keyphrase. The reciprocal of this specialization relation is expressed with the “is subtype of” keyphrase.

5.3.4 Collection relations

The Collection concept is a general mechanism for information organization. Collection relations are used to give a focus to particular subsets of the information in the STL description of a software application.

An information subset in a Collection is specified by naming other concept instances in an STL description. Collection hierarchies are described through aggregation relations.

There is no restriction on the number of Collections into which a concept instance may be grouped. Thus, Collections can support many viewpoints of the STL content as interlinked hierarchies of sets. Here are two common examples. A data dictionary groups all data definitions into a Collection. A data flow context diagram groups the top-level InternalAction with the ExternalActions and interconnecting ConnectionPaths into a Collection.

The Collection definition statement contains a usage label for indicating the intended interpretation of the Collection in either a standard or nonstandard way.

5.3.5 Presentation relations

Each instance of a software behavior concept may be mapped to an instance of a graphic image. For example, ConnectionPath df16 may be presented by curved arrow arc07. Collection concept instances may be mapped similarly, because they are often used for hierarchically structuring software category concept instances.

These concept definitions have the relation “is pictured with” for mapping a concept instance to an externally defined graphic image instance.

6. Specification of software behavior representation concepts

Each of the subclauses in this clause provides a textual description of an STL concept, followed by a concept definition template. Clause 5 describes the format for these concept definition templates. Although the concept definitions in this clause provide the normative description of the concepts for software behavior representation in this standard, informative Annex A contains entity-relationship diagrams that illustrate the major interrelationships between these concepts. The set of concept illustration diagrams in Annex A is organized to show subtypes and interrelationships, and it depicts the majority of these, but it does not provide a complete representation.

Each concept description begins with a summary table (not numbered) that contains the concept definition from Clause 3, a brief indication of the role or purpose of the concept for software behavior representations, and an indication of some of the more significant relationships with the other STL concepts, or other aspects of usage.

Concept	The name of the concept
Definition	What it is
Purpose	Why it is in the set of STL concepts
Usage	Some of the more significant descriptive uses for the concept

In the concept definition templates, the tables in this clause, a mnemonic name containing an underscore marks where an instance value is to be supplied for an attribute within a sentence written in the STL. A note below the section of the template with the mnemonic name indicates which value subtype is allowed in place of the mnemonic. Each table in Clause 6 follows the pattern established by Table 5 and illustrated by Table 6.

Within each table is a concept meaning entry. This entry can be considered as an amalgam of the entries in the brief summary table as described above. For continuity reasons, these entries are close to or identical to those used in IEEE Std 1175-1991 [B5].

6.1 Action

Concept	Action
Definition	A process of transformation that operates upon data or other types of inputs to create data, produce outputs, or change the state or condition of the subject software
Purpose	Provides a means to represent algorithms, computational procedures, and any other operations that a software application performs; used also to represent the methods offered by an object in the context of object-oriented analysis
Usage	Actions use and produce DataItems and cause StateTransitions

The Action concept represents the dynamic behavior of the subject software with respect to its environment. An Action concept sentence is used to describe this behavior in cause/effect form. It contains a possible relation for describing a method by which the behavior is produced in an InternalAction, but that description is not semantically interpreted.

In practice, the STL Action concept instances may be called transactions, processes, functions, services, methods, procedures, tasks, activities, or operations. This standard simply generalizes all of these to “Action.” An Action may be simple, compound, or other. Simple Actions transform control or data. Compound Actions transform both control and data. Other Actions are primarily resource manipulators.

Actions have inputs: DataItems used and EventItems received. Actions have context: States in which they can occur and preconditions to be met in order to occur. Actions have results: DataItems produced and EventItems transmitted. Actions have context effects: States changed and postconditions satisfied. Inputs and outputs are propagated through ConnectionPaths. Actions may have a decomposition structure. Multiple Actions may be performed by a single unit of code.

The Action concept has two subtypes: InternalAction and ExternalAction. InternalActions are those performed by the software. Because their accomplishment is the central purpose of the software, the concept sentence template includes additional properties, interactions, and relationships for them.

ExternalActions are those performed by the environment. They are described only to the extent of their interactions through ConnectionPaths with InternalActions or DataStores. ExternalActions represent any external “actors,” including other software components, organizations, networks, external data stores, and so on. Table 8 presents the concept definition for Action, and Table 9 presents the concept subtype definitions for Action.

Table 8—Action concept specification

concept name:	Action
concept meaning:	a transform, internal or external to the subject software , using and producing DataItems and causing StateTransitions
concept text presentation:	
sentence keyword:	Action
sentence identifier:	ActionID
possible attributes:	
has label	label_string
has description	description_string
is actiontype	internal external
possible relations:	
uses DataItem	DataItemID { , DataItemID }
produces DataItem	DataItemID { , DataItemID }
receives EventItem	EventItemID { , EventItemID }
transmits EventItem	EventItemID { , EventItemID }
is connected from ConnectionPath	ConnectionPathID { , ConnectionPathID }
is connected to ConnectionPath	ConnectionPathID { , ConnectionPathID }
has component Action	ActionID { , ActionID }
is component of Action	ActionID { , ActionID }
is grouped into Collection	CollectionID { , CollectionID }
is pictured with GraphicSymbol	GraphicSymbolID
notes:	
(1) The label_string is a stringValueShort.	
(2) The description_string is a stringValueLong.	
(3) Refer to Clause 7.5.5 for the value types.	
(4) The actiontype is a discriminator for the allowed subtypes of Action.	

Table 9—Action subtypes specification

subtype text presentation:		<i>for actiontype internal</i> (InternalAction)
possible attributes:		
is actiontype		internal
has process method		method_description
has transform purpose		control data compound
has criticality		mandatory desirable optional
has duration time units		time_units
has duration time minimum		min_duration
has duration time maximum		max_duration
possible relations:		
is allowed in State		StateID { , StateID }
causes StateTransition		StateTransitionID { , StateTransitionID }
acts only if Condition		ConditionID
satisfies Condition		ConditionID
is encapsulated in Object		ObjectID { , ObjectID }
is offered by Object		ObjectID { , ObjectID }
notes:		
(1) The method_description is a stringValueLong.		
(2) The time_units is a stringValueShort.		
(3) The min_duration is a realValue.		
(4) The max_duration is a realValue.		
(5) The method_description is often used to describe some procedural logic by which the Action can be accomplished.		
(6) The min_duration and max_duration are expressed in units of time_units.		
(7) The STL provides no explicit composition rules for components; rather, component Actions are defined with same-named input or output DataItems and different preconditions.		
subtype text presentation:		<i>for actiontype external</i> (ExternalAction)
possible attributes:		
is actiontype		external
possible relations:		
none		

6.2 Collection

Concept	Collection
Definition	An unrestricted grouping of software behavior concept instances into a particular named subset
Purpose	Provides a means to record context-dependent groupings of data or processes, such as the data contained in a report or displayed on a graphical user interface
Usage	Collections gather other selected STL concept instances into an identified grouping

Instances of the Collection concept are used to focus on particular subsets of information in the software behavior description. The subset is specified by reference to other concept instance names defined in the software behavior description. A usage label indicates the intended use of the Collection in a standard or nonstandard way. Standard usage keyword labels for Collections are dfd (for data flow models), erd (for entity-relationship models), std (for state transition models), and designunit (for design allocations). Table 10 presents the concept definition for Collection.

6.3 Condition

Concept	Condition
Definition	A true or false logical predicate
Purpose	Provides a means to control the Actions of the subject software
Usage	Logically evaluates values of DataItems and Constants to characterize a situation within the subject software

The Condition concept represents the ability of the subject software to apply tests for equality and ordering among DataItems and Constants. The logical result serves two purposes. It can serve as a precondition to distinguish which Actions can be performed for certain inputs, or it can serve as a postcondition to make assertions about the results of an Action. The sentence defines the logical value of a Condition by primitive expressions or by logical expressions that are constructed from primitive expressions.

The Condition concept permits logical conditions to be identified, named, and referenced by an identifier. A “condExpression” is an STL syntactic element used for embedding a logical condition (without an identifier) into an STL sentence. Because a condExpression contains embedded IDs for other STL concepts, it establishes a set of relations between Condition and the concepts referenced. The inverse of these relations is the “is component of Condition” relation. This inverse relation provides cross-reference pointers to the usage of Conditions within condExpressions, but it (unlike the corresponding condExpression) does not provide sufficient information to be evaluated as a logical value.

Conditions do not have a default equivalence with a Boolean DataItem. Unlike a Condition, a Boolean DataItem cannot be used in the “acts only if” or “satisfies” possible relations of an Action. Unlike a Boolean DataItem, a Condition cannot be stored in a DataStore or propagated through a ConnectionPath.

Table 11 presents the concept definition for Condition.

Table 10—Collection concept specification

concept name:	Collection
concept meaning:	an unrestricted grouping of concept instances into a particular named subset
concept text presentation:	
sentence keyword:	Collection
sentence identifier:	CollectionID
possible attributes:	
has label	label_string
has description	description_string
has standard usage	dfd erd std designunit
has external usage	usage_explanation
possible relations:	
has component Collection	CollectionID { , CollectionID }
is component of Collection	CollectionID { , CollectionID }
groups Action	ActionID { , ActionID }
groups Condition	ConditionID { , ConditionID }
groups ConnectionPath	ConnectionPathID { , ConnectionPathID }
groups DataItem	DataItemID { , DataItemID }
groups DataKey	DataKeyID { , DataKeyID }
groups DataPart	DataPartID { , DataPartID }
notes:	
(1) The label_string is a stringValueShort.	
(2) The description_string is a stringValueLong.	
(3) The usage_explanation is a stringValue.	
(4) Refer to 7.5.5 for the value types.	
(5) The intended usage of the Collection is indicated by the presence of either the “has standard usage” attribute or the “has external usage” attribute, but not both.	
(6) A Collection can be structured from other Collections as described by the two “component” possible relations, so other Collection identifiers cannot be included in the “groups” possible relation.	
(7) The STL provides no explicit composition rules for “components.”	

Table 11—Condition concept specification

concept name:	Condition
concept meaning:	an unrestricted grouping of concept instances into a particular named subset
concept text presentation:	
sentence keyword:	Condition
sentence identifier:	ConditionID
possible attributes:	
has label	label_string
has description	description_string
possible relations:	
is true if	condExpression
permits Action	ActionID { , ActionID }
is satisfied by Action	ActionID { , ActionID }
establishes State	StateID { , StateID }
is component of Condition	ConditionID { , ConditionID }
is encapsulated in Object	ObjectID { , ObjectID }
is grouped into collection	CollectionID { , CollectionID }
is pictured with GraphicSymbol	GraphicSymbolID
notes:	
(1) The label_string is a stringValueShort.	
(2) The description_string is a stringValueLong.	
(3) Refer to 7.5.5 for the value types.	
(4) The “is true if” possible relation departs from the convention for relations; it does not end with a concept name because any number of DataItems, Constants, and other Conditions may be involved in the condExpression.	
(5) The condExpression is recorded as a stringValueLong, and in this respect is treated as if an attribute. However, the contents of this string, as explained in 7.5.6, establish a set of relations.	
(6) There is no separate “has component” possible relation because any component Condition appears by name in the condExpression.	

6.4 ConnectionPath

Concept	ConnectionPath
Definition	A pathway for the propagation of information
Purpose	Provides the means to state requirements for, identify constraints on, or otherwise characterize the physical and timing aspects of the movement of data between components within the subject software, and between the subject software and external actors
Usage	Characterizes the movement of DataItems and EventItems between Actions, or between Actions and DataStores

The ConnectionPath concept expresses the propagation of information from a source to a destination.

The ConnectionPath concept has two subtypes, DataConnectionPath and EventConnectionPath, which are distinguished principally by the kind of information that they propagate.

DataConnectionPaths propagate DataItems. Propagation can be from Action to a DataStore, from a DataStore to an Action, or from one Action to another.

EventConnectionPaths propagate EventItems (markers of time). Propagation can be from one Action to another.

Merge and split relations are used to relate the sources, destinations, and contents of one ConnectionPath to those of other ConnectionPaths. Here are some examples of such relations:

- A DataItem with one source may be carried to one destination.
- Several DataItems, each from its own source, may be carried on separate ConnectionPaths and merged into the contents of one ConnectionPath to one destination.
- A DataItem carried on one ConnectionPath may be split and duplicates of it carried on several ConnectionPaths to several destinations.
- A DataItem carried on one ConnectionPath may be split and various substructures and components carried on several ConnectionPaths to several destinations.
- Several different DataItems, each from its own source and carried on separate ConnectionPaths, are merged, and a superstructure or aggregate is carried on one ConnectionPath to one destination.
- Several identical DataItems, each from its own source and carried on separate ConnectionPaths, are merged, and a single copy of it carried on one ConnectionPath to one destination.

There are no assumptions concerning hierarchical inclusion of sources and destinations. That is, splits and merges are used to document continuity, whether between parent/child diagrams or between multiple source/destination nodes in a network.

Normally, connectivity can be determined by matching ConnectionPath contents across splits and merges between a source and a destination. Contents must be accountable by reference to the DataType/EventType hierarchies of which the DataItems/EventItems are instances. Where it is useful to emphasize a continuing presence of a particular group of items in several ConnectionPaths, a separate ConnectionPath can be defined and related to the others with the “has component/is component of” possible relations.

Table 12 presents the concept definition for ConnectionPath, and Table 13 presents the concept subtype definitions for ConnectionPath.

Table 12—ConnectionPath concept specification

concept name:	ConnectionPath
concept meaning:	a path for the propagation of information between Actions or between Actions and DataStores
concept text presentation:	
sentence keyword:	ConnectionPath
sentence identifier:	ConnectionPathID
possible attributes:	
has label	label_string
has description	description_string
has time units	time_units
has bandwidth minimum	min_bandwidth
has bandwidth maximum	max_bandwidth
has delay time minimum	min_delay
has delay time maximum	max_delay
has transfer timing	asynchronous synchronous
is connectiontype	data event
possible relations:	
connects from Action	ActionID
connects to Action	ActionID
is split from ConnectionPath	ConnectionPathID
is merged from ConnectionPath	ConnectionPathID { , ConnectionPathID }
splits into ConnectionPath	ConnectionPathID { , ConnectionPathID }
merges into ConnectionPath	ConnectionPathID
has component ConnectionPath	ConnectionPathID { , ConnectionPathID }
is component of ConnectionPath	ConnectionPathID { , ConnectionPathID }
is bidirectional with ConnectionPath	ConnectionPathID
is encapsulated in Object	ObjectID { , ObjectID }
is grouped into Collection	CollectionID { , CollectionID }
is pictured with GraphicSymbol	GraphicSymbolID

Table 12—ConnectionPath concept specification (continued)

notes:
(1) The label_string is a stringValueShort.
(2) The description_string is a stringValueLong.
(3) The time_units is a stringValueShort.
(4) The min_bandwidth is a realValue.
(5) The max_bandwidth is a realValue.
(6) The min_delay is a realValue.
(7) The max_delay is a realValue.
(8) Refer to 7.5.4 and 7.5.5 for a discussion of the value types and value literals.
(9) The connectiontype is a discriminator for the allowed subtypes of ConnectionPath.
(10) The bandwidth values are expressed in units of repetitions/time_unit.
(11) The delay values are expressed in units of time_unit.
(12) The “bidirectional” relation is used to specify two distinct ConnectionPaths with opposing directions as the paired components of a bidirectional ConnectionPath.

Table 13—ConnectionPath subtypes specification

subtype text presentation:	<i>for connectiontype data (DataConnectionPath)</i>
possible attributes:	
is connectiontype	data
has flow characteristic	discrete continuous
has retrieval effect	destructive nondestructive
possible relations:	
carries DataItem	DataItemID { , DataItemID }
connects from DataStore	DataStoreID
connects to DataStore	DataStoreID
notes:	
(1) Only data ConnectionPaths can connect to DataStores.	
(2) Either “connects from Action” or “connects from DataStore” can be used, but not both.	
(3) Either “connects to Action” or “connects to DataStore” can be used, but not both.	
subtype text presentation:	<i>for connectiontype event (EventConnectionPath)</i>
possible attributes:	
is connectiontype	event
possible relations:	
carries EventItem	EventItemID { , EventItemID }
notes:	
(1) Event ConnectionPaths can connect only to Actions.	

6.5 Constant

Concept	Constant
Definition	A reference name for a numerical, lexical, or externally specified fixed (unchanging) value
Purpose	Provides a means to refer to a known value by name rather than by embedding a literal in the STL
Usage	Constants may be associated with DataItems and DataParts, and may appear in condExpressions

The Constant concept provides an alternative to literals for expressing values. It has several descriptive advantages over literals.

- A Constant instance definition provides a symbolic name that can be used throughout a software behavior description for referring to the Constant's defined value.
- A Constant can be given a descriptive name to indicate its meaning in the subject specification.
- A Constant can have description text attached to the definition of its name.

For each subtype of value, there is a corresponding subtype of Constant. All subtypes of Constant, except the “external” subtype, correspond with subtypes of literals in the syntax definition for the STL. The “external” subtype is used to avoid the imprecise definition of values such as transcendental numbers. There is no restriction on names of Constants because no Constants are predefined in the STL.

Table 14 presents the concept definition for Constant, and Table 15 presents the concept subtype definitions for Constant.

Table 14—Constant concept specification

concept name:	Constant
concept meaning:	a reference for a numerical, lexical, or externally specified literal
concept text presentation:	
sentence keyword:	Constant
sentence identifier:	ConstantID
possible attributes:	
has label	label_string
has description	description_string
is value type	integer real character string stringshort stringlong structured external
possible relations:	
specifies DataPart	DataPartID { , DataPartID }
instantiates DataItem	DataItemID { , DataItemID }
is pictured with GraphicSymbol	GraphicSymbolID
notes:	
(1) The label_string is a stringValueShort.	
(2) The description_string is a stringValueLong.	
(3) Refer to 7.5.4 and 7.5.5 for a discussion of the value types and value literals.	
(4) The valuetype is a discriminator for the allowed subtypes of Constant.	

Table 15—Constant subtype specification

subtype text presentation: <i>for valuetype integer (IntegerConstant)</i> possible attributes: is valuetype integer has value i_value possible relations: none notes: (1) The i_value is an integerValue.
subtype text presentation: <i>for valuetype real (RealConstant)</i> possible attributes: is valuetype real has value r_value possible relations: none notes: (1) The r_value is a realValue.
subtype text presentation: <i>for valuetype character (CharacterConstant)</i> possible attributes: is valuetype character has value c_value possible relations: none notes: (1) The c_value is a characterValue.
subtype text presentation: <i>for valuetype string (StringConstant)</i> possible attributes: is valuetype string has value s_value possible relations: none notes: (1) The s_value is a stringValue.

Table 15—Constant subtype specification (continued)

subtype text presentation:	<i>for valuetype stringshort</i> (StringshortConstant)
possible attributes:	
is valuetype	stringshort
has value	s_valueshort
possible relations:	
none	
notes:	
(1) The s_valueshort is a stringValueShort.	

subtype text presentation:	<i>for valuetype stringlong</i> (StringlongConstant)
possible attributes:	
is valuetype	stringlong
has value	s_valuelong
possible relations:	
none	
notes:	
(1) The s_valuelong is a stringValueLong.	

subtype text presentation:	<i>for valuetype structured</i> (StructuredConstant)
possible attributes:	
is valuetype	structured
has value	m_value
possible relations:	
none	
notes:	
(1) The m_value is a structuredValue.	
(2) Matched left and right parentheses are used in structuredValue to indicate substructure nesting.	

subtype text presentation:	<i>for valuetype external</i> (ExternalConstant)
possible attributes:	
is valuetype	external
has external value	definition_reference
possible relations:	
none	
notes:	
(1) The definition_reference is a stringValue.	
(2) Communicating tools will establish standard definition_reference strings for Constants such as e (Naperian logarithm base), pi, etc.	

6.6 DataItem

Concept	DataItem
Definition	A possible value or structure of values that can be retained or processed
Purpose	Provides a means for variables to be represented in STL
Usage	DataItems may be used and produced by Actions, or stored by the subject software, and may have a State

The DataItem concept represents any one instance of a DataType in a specific context. It may be input to or output from an Action, it may be supplied from or accepted into a DataStore, or it may be held in a DataStore. DataItems may be carried by a ConnectionPath or a network of ConnectionPaths from one Action to another Action, from an Action to a DataStore, or from a DataStore to an Action.

Table 16 presents the concept definition for DataItem.

6.7 DataKey

Concept	DataKey
Definition	An identifier for a particular grouping of data
Purpose	Provides a means to represent table keys or entity keys as used in traditional database design logical models
Usage	DataKeys provide a means to identify (select) a particular entity DataType, which is the STL equivalent to an entity as used in traditional entity-relationship-attribute models. DataKeys are defined by referencing a DataPart whose value is the key value

The DataKey concept, together with the DataType and DataPart concepts, provides the means for designating the identifier attribute(s) in an entity definition. It is used similarly to designate the identifier components in the rows of a table definition.

A DataKey for a DataType is defined by referencing the DataPart(s) whose value(s) serve uniquely to identify different instances of the DataType. If only one DataPart value is needed, the key is called “simple.” When values of more than one DataPart are needed, the key is called “composite.”

When a DataType has multiple DataKeys, they are commonly labeled with names such as “primary” or “alternate.” If a DataType has DataParts whose instances serve to identify instances of another DataType, the key is called “foreign.”

Table 17 presents the concept definition for DataKey.

Table 16—DataItem concept specification

concept name:	DataItem
concept meaning:	a possible value or structure of values that can be retained or processed
concept text presentation:	
sentence keyword:	DataItem
sentence identifier:	DataItemID
possible attributes:	
has label	label_string
has description	description_string
has format	data_format
has units	data_units
has placeholder value	p_value
has null occurrences	allowed disallowed
possible relations:	
is an instance of DataType	DataTypeID
is an instance of Constant	ConstantID
is used by Action	ActionID { , ActionID }
is produced by Action	ActionID { , ActionID }
is characterized by State	StateID { , StateID }
is carried by ConnectionPath	ConnectionPathID { , ConnectionPathID }
is supplied by DataStore	DataStoreID { , DataStoreID }
is accepted by DataStore	DataStoreID { , DataStoreID }
is stored by DataStore	DataStoreID { , DataStoreID }
is encapsulated in Object	ObjectID { , ObjectID }
is offered by Object	ObjectID { , ObjectID }
is grouped into Collection	CollectionID { , CollectionID }
is pictured with GraphicSymbol	GraphicSymbolID

Table 16—DataItem concept specification *(continued)*

notes: (1) The label_string is a stringValueShort. (2) The description_string is a stringValueLong. (3) The data_format is a stringValue. (4) The data_units is a stringValueShort. (5) The p_value is a stringValue. (6) Refer to 7.5.4 and 7.5.5 for a discussion of the value types and value literals. (7) If null occurrences are allowed for a DataItem, this indicates that information may be missing at the places where the DataItem is mentioned. (8) Format or units specifications here supersede those in the referenced DataType. (9) Either the “has placeholder value” possible attribute or one of the “is an instance of” possible relations may be used, but not more than one. (10) The “has placeholder value” possible attribute allows a textual description in place of an actual domain or structure specification.
--

Table 17—DataKey concept specification

concept name:	DataKey
concept meaning:	a reference to one or more DataParts whose value(s) serve to uniquely identify different instances of a DataType
concept text presentation:	
sentence keyword:	DataKey
sentence identifier:	DataKeyID
possible attributes:	
has label	label_string
has description	description_string
has identifier purpose	primary alternate foreign
possible relations:	
references DataPart	DataPartID { , DataPartID }
provides access to DataType	DataTypeID { , DataTypeID }
shares instances with DataKey	DataKeyID { , DataKeyID }
is grouped into Collection	CollectionID { , CollectionID }
is pictured with GraphicSymbol	GraphicSymbolID
notes:	
(1) The label_string is a stringValueShort.	
(2) The description_string is a stringValueLong.	
(3) Refer to 7.5.4 and 7.5.5 for a discussion of the value types and value literals.	
(4) The referenced DataParts must be components of the DataType.	
(5) The DataTypeID must be of datatypeclass table or entity.	
(6) An identifier purpose of “primary” means the key uniquely identifies one instance of the DataType and is the “preferred” identifier for implementation.	
(7) An identifier purpose of “alternate” means the key uniquely identifies one instance of the DataType.	
(8) An identifier purpose of “foreign” means the key does not uniquely identify one instance of the subject DataType, but it does uniquely identify one instance of some other DataType. Such overlapping of occurrences of a foreign key in one DataType with the primary or alternate key in another DataType is the basis of the “join” operation in the relational data model. The “shares instances” possible relation relates those keys having such overlapping instances.	

6.8 DataPart

Concept	DataPart
Definition	A component of a structured DataType with a domain specified by reference to some other DataType
Purpose	Provides a means to represent attributes as used in traditional database design logical models
Usage	A DataPart is a component of an entity DataType, which is the STL equivalent to an entity as used in traditional entity-relationship-attribute models

The DataPart concept, together with the DataType concept, provides a way for describing structures of values.

An instance of the DataPart concept is a named substructure of a structured DataType. The DataPart instance definition specifies the domain of the component as some other DataType. That DataType itself may be structured.

A DataType can appear as one or more components of other DataTypes, but each appearance has a unique DataPart name for identification and reference. DataPart names are labels by which individual components in an instance of the structure are identified. Separating the description of a data structure between DataTypes and DataParts allows separating the specification of a data domain (as a DataType) from identifying a structure component whose values can be occurrences in that domain.

Here is an example to illustrate this difference. Assume a DataType named State-Abbreviation, which is defined as the list of stringValues “AK,” “AL,” ... , “WY.” For a purchase invoice, say, showing a shipping address and a billing address, there is both a Ship_State component and a Bill_State component. These are two different components, or DataParts, of the Invoice DataType. The definitions of each would reference the State-Abbreviation DataType as their domain of values.

Table 18 presents the concept definition for DataPart.

Table 18—DataPart concept specification

concept name:	DataPart
concept meaning:	a component of a structured DataType with a domain specified by reference to some other DataType
concept text presentation:	
sentence keyword:	DataPart
sentence identifier:	DataPartID
possible attributes:	
has label	label_string
has description	description_string
has format	data_format
has units	data_units
has placeholder value	p_value
has null occurrences	allowed disallowed
possible relations:	
references DataPart	DataTypeID
is specified as Constant	ConstantID
shares instances with DataKey	DataKeyID { , DataKeyID }
is component of DataType	DataTypeID { , DataTypeID }
is component of DataRole	DataRoleID { , DataRoleID }
is grouped into Collection	CollectionID { , CollectionID }
is pictured with GraphicSymbol	GraphicSymbolID
notes:	
(1) The label_string is a stringValueShort.	
(2) The description_string is a stringValueLong.	
(3) The data_format is a stringValue.	
(4) The data_units is a stringValueShort.	
(5) The p_value is a stringValue.	
(6) Refer to 7.5.4 and 7.5.5 for a discussion of the value types and value literals.	
(7) If null occurrences are allowed in the DataPart, this indicates that information may be missing in instances of those DataTypes where the DataPart is a component.	
(8) Format or units specifications here supersede those in the specifying DataType.	
(9) Either the “has placeholder value” possible attribute or one of the “is specified as” possible relations can be used, but not more than one.	
(10) The “has placeholder value” possible attribute allows a textual description in place of an actual domain or structure specification.	

6.9 DataRole

Concept	DataRole
Definition	A characterization of the way an entity DataType participates in a relationship
Purpose	Provides a means to represent relationships as used in traditional database design logical models
Usage	A DataRole is played by a DataType with respect to one or more other DataTypes, and it can also represent an attributed relationship by virtue of having component DataParts

The DataRole concept, together with the entity and relationship DataType concepts, provides the means for specifying how instances of entity DataTypes are associated in the application.

Entity DataTypes are specified as participants in a relationship with a DataRole. The DataRole is characterized by the entity DataType whose instances are involved in the relationship and the cardinality of that involvement.

When a relationship has more than two DataRoles involved with it (an N-ary relationship), relationship instances may occur differently from the entity or relationship instances playing the roles. For this reason, DataRoles have both an “inner” cardinality and an “outer” cardinality specification. The inner cardinality specifies the occurrences of a relationship for each occurrence of the role participant. The outer cardinality specifies the occurrences of role participants for each occurrence of the relationship.

Table 19 presents the concept definition for DataRole.

6.10 DataStore

Concept	DataStore
Definition	A specification of data retention capabilities for the subject software
Purpose	Provides a means to represent the retention of data controlled by the subject software
Usage	A DataStore can accept, hold, and supply DataItems as in response to requests by a software process

The DataStore concept expresses the retention of information from one time to another. DataStores allow Actions of the software to retain information acquired at one time, or to accumulate information over time, and then to reference it at a later time. DataStores are characterized as depletable if reference to their content causes loss of their content. If references are not destructive, the DataStore is nondepletable. Only instances of data can be retained. Information about the occurrence of an Event can be retained (e.g., has it occurred, or when did it occur), but the Event occurrence itself cannot be retained. For this reason, the STL does not have an “EventStore” concept. Table 20 presents the concept definition for DataStore.