Date: May 05, 2004

To: T10 Committee (SCSI)

From: George Penokie (IBM/Tivoli)

Subject: SAM-3: Converting to UML part 1

1 Overview

The current SCSI architecture follows no particular documentation convention outside of it's own T10 invented one. This proposal suggests that the documenting convention be changed to comply with the unified modeling language (UML) standards. UML is widely used and, as such, would make the concepts defined in SAM-3 more understandable to a larger audience.

For those interested in learning about UML I would suggest the following reading materials:

a) SAMS Teach Yourself UML in 24 hours (This is a kind of crash course that will give you the basics);
b) The Unified Modeling Language User Guide by Grady Booch, et al. (This is considered the UML bible by many);
c) The Unified Modeling Language Reference Manual by James Rumbaugh, et al. (This is considered the UML bible part 2 by many); and
d) UML Distilled Second Edition A brief Guide to the Standard Object Modeling Language by Martin Fowler (This is a good reference book).

There are many more and there is no lack of opinion as to which is most useful. Also, if you are a real masochist you can read the standard. Go to http://www.omg.org/cgi-bin/apps/doclist.pl web site and search on UML. What you are looking for is the Unified Modeling Language: OCL version 2.0 ptc/03-08-08.

2 The first step - class vs object

The first thing that needs to be done to accomplish this conversion is to understand that most of the things SAM-3 currently calls objects are not objects but classes.

Classes are defined in UML as:

2.0.1 class: A description of a set of objects that share the same attributes, operations, relationships, and semantics. Classes are used to represent software things, hardware things, and thngs that are pruely conceptual. Every class has a class name. Classes may have attributes and may support operations.

Objects are defined in UML as:

2.0.2 object: An entity with a well-defined boundary and identity that encapsulates state and behavior. All objects are instances of classes (i.e., a concrete manifestation of a class is an object).

Some supporting UML terminology:

2.0.3 attributes: A named property of a class that describes the range of values that the class or its instances (i.e., objects) may hold.

2.0.4 behavior: How an object acts and reacts to requests from other objects.

2.0.5 class name: A lable of a class.

2.0.6 constraint: A mechanism for specifying semantics or conditions that are maintained as true between entities (e.g., a required condition between associations).

2.0.7 operation: A service that may be requested from any object of the class in order to effect behavior. Operations describe what a class is allowed to do and may be a request or a question. A request may change the state of the object but a question should not.

2.0.8 state: One of the possible conditions in which an object may exist that normally changes over time and is represented by attributes.

For a non-SCSI example of classes and objects see figure 1. For a SCSI example of classes and objects see figure 2.

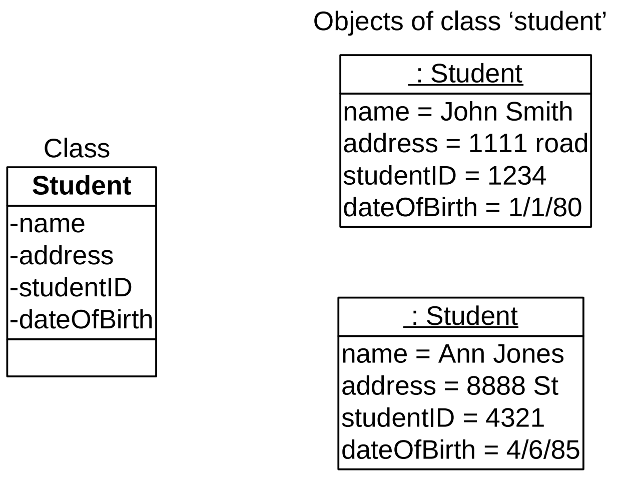


Figure 1 — NonSCSI example of class and object representation

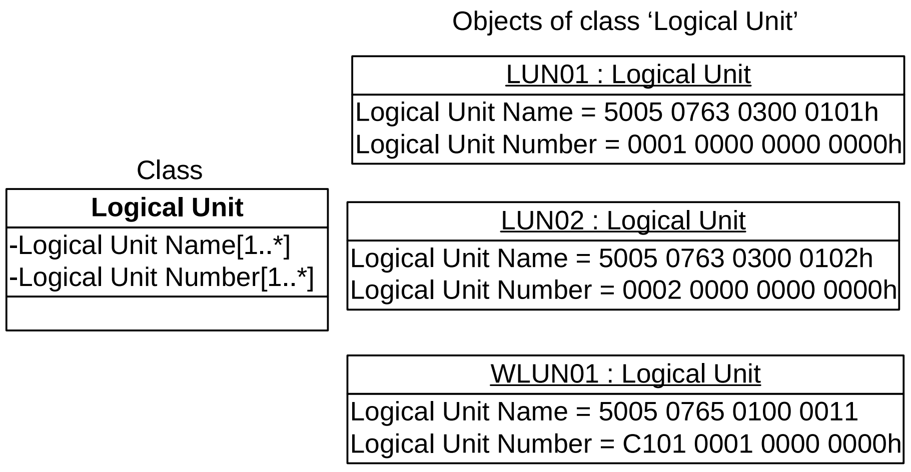

Objects of class ‘Logical Unit’
LUN01 : Logical Unit
Logical Unit Name = 5005 0763 0300 0101h
Logical Unit Number = 0001 0000 0000 0000h
LUN02 : Logical Unit
Logical Unit Name = 5005 0763 0300 0102h
Logical Unit Number = 0002 0000 0000 0000h
WLUN01 : Logical Unit
Logical Unit Name = 5005 0765 0100 0011
Logical Unit Number = C101 0001 0000 0000h
Class
Logical Unit
-Logical Unit Name[1..*]
-Logical Unit Number[1..*]

Figure 2 — SCSI example of class and object representation

3 Example of a class diagram

For an example of what figures in SAM-3 would look like a converted logical unit model (see figure 3) as currently shown in SAM-3 would look like what is shown in figure 4.

.

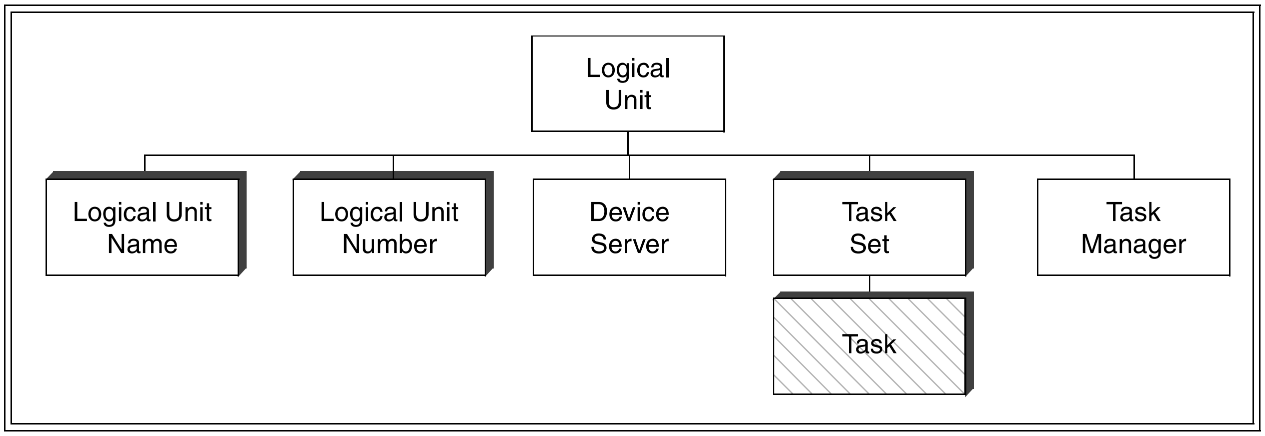


Figure 3 — Logical unit model

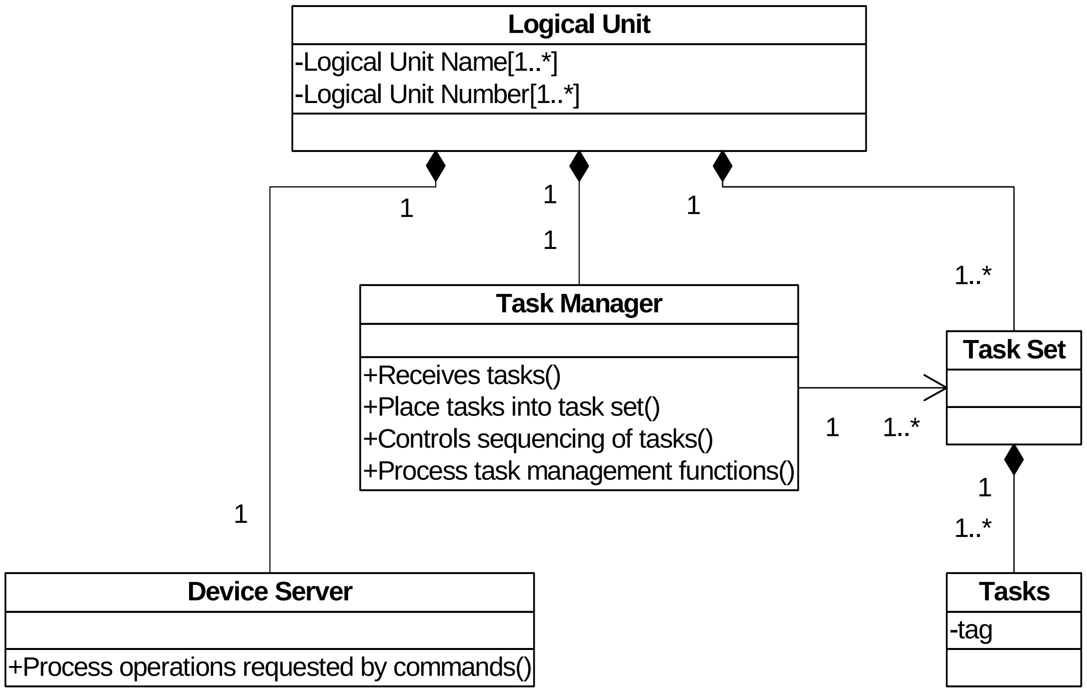

Logical Unit
-Logical Unit Name[1..*]
-Logical Unit Number[1..*]
1
1
1
1
1..*
Task Manager
+Receives tasks()
+Place tasks into task set()
+Controls sequencing of tasks()
+Process task management functions()
1
1..*
Task Set
1
1..*
1
Device Server
+Process operations requested by commands()
Tasks
-tag

Figure 4 — Logical unit class diagram

1 SAM-3 changes

2 Definitions, symbols, abbreviations, and conventions

2.1 Definitions

2.1.1 ACA command: A command performed by a task with the ACA attribute (see 2.1.8, 4.11, and 8.6.5).

2.1.2 additional sense code: A combination of the ADDITIONAL SENSE CODE and ADDITIONAL SENSE CODE QUALIFIER fields in the sense data (see 2.1.117 and SPC-2).

2.1.3 aggregation: When used in class diagrams, aform of association that defines a whole-part relationship between the whole (i.e., aggregate) and its parts.

2.1.4 application client: A client that is the source of SCSI commands.

2.1.5 argument: A datum provided as input to or output from a procedure call (see 2.1.85).

2.1.6 association: When used in class diagrams arelationship between two or more classes that specifies connections among thier objects (i.e., relationship that specifies that objects of one class are connected to objects of another class).

2.1.7 attribute: A named property of a class that describes the range of values that the class or its objects may hold.

2.1.8 auto contingent allegiance (ACA): The task set condition established following the return of a CHECK CONDITION status when the NACA bit is set to one in the CONTROL byte. See 5.9.2.

2.1.9 background operation: An operation started by a command that continues processing after the task containing the command is no longer in the task set. See 5.5.

2.1.10 blocked task state: When in this state a task is prevented from completing due to an ACA condition.

2.1.11 blocking boundary: A task set boundary denoting a set of conditions that inhibit tasks outside the boundary from entering the enabled task state.

2.1.12 byte: An 8-bit construct.

2.1.13 client-server: A relationship established between a pair of distributed entities where one (the client) requests the other (the server) to perform some operation or unit of work on the client's behalf.

2.1.14 client: An entity that requests a service from a server. This standard defines one client, the application client.

2.1.15 code value: A defined numeric value, possibly a member of a series of defined numeric values, representing an identified and described instance or condition. Code values are defined to be used in a specific field (see 2.1.41), in a procedure call input argument (see 2.6.2), in a procedure call output argument, or in a procedure call result.

2.1.16 command: A request describing a unit of work to be performed by a device server.

2.1.17 command descriptor block (CDB): A structure used to communicate a command from an application client to a device server. A CDB may have a fixed length of up to 16 bytes or a variable length of between 12 and 260 bytes.

2.1.18 command standard: A SCSI standard that defines the model, commands, and parameter data for a device type (e.g., SPC-3, SBC-2, SSC-2, SMC-2, MMC-3, or SES-2). See clause 1.

2.1.19 completed command: A command that has ended by returning a status and service response of TASK COMPLETE or LINKED COMMAND COMPLETE.

2.1.20 completed task: A task that has ended by returning a status and service response of TASK COMPLETE. The actual events comprising the TASK COMPLETE response are SCSI transport protocol specific.

2.1.21 confirmation: A response returned to an application client or device server that signals the completion of a service request.

2.1.22 confirmed SCSI transport protocol service: A service available at the SCSI transport protocol service interface that includes a confirmation of completion.

2.1.23 constraint A mechanism for specifying semantics or conditions that are maintained as true between entities (e.g., a required condition between associations).

2.1.24 class: A description of a set of objects that share the same attributes, operations, relationships, and semantics. Classes may have attributes and may support operations.

2.1.25 current task: A task that has a data transfer SCSI transport protocol service request in progress (see 5.4.3) or is in the process of sending command status. Each SCSI transport protocol standard should define the SCSI transport protocol specific conditions under which a task is considered a current task.

2.1.26 deferred error: An error generated by a background operation (see 2.1.9).

2.1.27 dependency: When used in a class diagram, a relationship between two classes where a change to one class (i.e., the independent class) may cause a change in the other class (i.e., the dependent class).

2.1.28 dependent logical unit: A logical unit that is addressed via some other logical unit(s) in a hierarchical logical unit structure (see 2.1.45), also a logical unit that is at a higher numbered level in the hierarchy than the referenced logical unit (see 4.14).

2.1.29 device identifier: A term used by previous versions of this standard (see Annex B).

2.1.30 device model: The description of a type of SCSI target device (e.g., block, stream).

2.1.31 device server: A class within a logical unit that processes SCSI tasks according to the requirements for task management described in clause 8.

2.1.32 device service request: A request submitted by an application client conveying a SCSI command to a device server.

2.1.33 device service response: The response returned to an application client by a device server on completion of a SCSI command.

2.1.34 domain: An I/O system consisting of a set of SCSI devices that interact with one another by means of a service delivery subsystem.

2.1.35 dormant task state: When in this state a task is prevented from entering the enabled task state (see 2.1.36) due to the presence of certain other tasks in the task set.

2.1.36 enabled task state: When in this state a task may complete at any time or is waiting to receive the next command in a series of linked commands.

2.1.37 faulted initiator port: The SCSI initiator port to which a CHECK CONDITION status was returned that resulted in the establishment of an ACA. The faulted initiator port condition is cleared when the ACA condition is cleared.

2.1.38 faulted task set: A task set that contains a faulting task. The faulted task set condition is cleared when the ACA condition resulting from the CHECK CONDITION status is cleared.

2.1.39 faulting command: A command that completed with a status of CHECK CONDITION that resulted in the establishment of an ACA.

2.1.40 faulting task: A task that has completed with a status of CHECK CONDITION that resulted in the establishment of an ACA.

2.1.41 field: A group of one or more contiguous bits, part of a larger structure such as a CDB (see 2.1.17) or sense data (see 2.1.117).

2.1.42 function complete: A logical unit response indicating that a task management function has finished. The events comprising this response are SCSI transport protocol specific.

2.1.43 generalization: When used in class diagrams, a relationship among classes where one class (i.e., superclass) shares the attributes and/or operations on one or more classes (i.e., subclasses).

2.1.44 hard reset: A response to a power on (see 6.3.1) or reset (see 6.3.2) condition in which the SCSI device performs the operations described in 6.3.2.

2.1.45 hierarchical logical unit: An inverted tree structure for forming and parsing logical unit numbers (see 2.1.70) containing up to four addressable levels (see 4.14).

2.1.46 I_T nexus: A nexus between a SCSI initiator port and a SCSI target port (see 4.12).

2.1.47 I_T nexus loss: A condition resulting from a hard reset condition (see 6.3.2) in SCSI initiator devices or from delivery of an I_T nexus loss notification indication (see 6.4) in all SCSI devices in which the SCSI device performs the operations described in 6.3.4.

2.1.48 I_T nexus loss notification: An indication (see 6.4) from the SCSI transport protocol to the SCSI application layer that an I_T nexus no longer exists.

2.1.49 I_T_L nexus: A nexus between a SCSI initiator port, a SCSI target port, and a logical unit (see 4.12).

2.1.50 I_T_L_Q nexus: A nexus between a SCSI initiator port, a SCSI target port, a logical unit, and a task (see 4.12).

2.1.51 I_T_L_Q nexus transaction: The information transferred between SCSI ports in a single data structure with defined boundaries (e.g., an information unit).

2.1.52 I_T_L_x nexus: Either an I_T_L nexus or an I_T_L_Q nexus (see 4.12).

2.1.53 I/O operation: An operation defined by an unlinked SCSI command, a series of linked SCSI commands or a task management function.

2.1.54 implementation specific: A requirement or feature that is defined in a SCSI standard but whose implementation may be specified by the system integrator or vendor.

2.1.55 initiator: A term used by previous versions of this standard (see Annex B).

2.1.56 initiator device name: A SCSI device name of a SCSI initiator device (see 4.7.1).

2.1.57 initiator identifier: A term used by previous versions of this standard (see Annex B).

2.1.58 initiator port identifier: A value by which a SCSI initiator port is referenced within a domain. See 4.7.1.

2.1.59 initiator port name: A SCSI port name (see 2.1.106) of a SCSI initiator port or of a SCSI target/initiator port when operating as a SCSI initiator port. See 4.7.1.

2.1.60 interconnect subsystem: One or more interconnects that appear as a single path for the transfer of information between SCSI devices in a domain.

2.1.61 in transit: Information that has been sent to a remote entity but not yet received.

2.1.62 implicit head of queue: An optional processing model for specified commands wherein the specified commands may be treated as if they had been received with a HEAD OF QUEUE task attribute. See 8.2.

2.1.63 layer: A subdivision of the architecture constituted by SCSI initiator device and SCSI target device elements at the same level relative to the interconnect.

2.1.64 linked CDB: A CDB with the LINK bit in the CONTROL byte set to one.

2.1.65 linked command: One in a series of SCSI commands processed by a single task that collectively make up a discrete I/O operation. In such a series, each command is represented by the same I_T_L_Q nexus, and all, except the last, have the LINK bit in the CDB CONTROL byte set to one.

2.1.66 logical unit: A class within SCSI target device, containing a device server and task manager, that implements a device model and manages tasks to process SCSI commands sent by an application client. See 4.8.

2.1.67 logical unit reset: A logical unit response to a logical unit reset event in which the logical unit performs the operations described in 6.3.3.

2.1.68 logical unit reset event: An event that triggers a logical unit reset (see 2.1.67). Logical unit reset events include processing the LOGICAL UNIT RESET task management function (see 7.6) and hard reset (see 6.3.2).

2.1.69 logical unit inventory: The list of the logical unit numbers reported by a REPORT LUNS command (see SPC-2).

2.1.70 logical unit number (LUN): A 64-bit identifier for a logical unit.

2.1.71 media information: Information stored within a SCSI device that is non-volatile (retained through a power cycle) and accessible to a SCSI initiator device through the processing of SCSI commands.

2.1.72 multiplicity: A indication of the range of allowable instances that a class or an attribute may have.

2.1.73 name: A label of an object that is unique within a specified context and should never change (e.g., the term name and world wide identifier (WWID) may be interchangeable).

2.1.74 nexus: A relationship between two SCSI devices, and the SCSI initiator ports and SCSI target ports within those SCSI devices. See 4.12.

2.1.75 non-faulted initiator port: A SCSI initiator port that is not a faulted initiator port (see 2.1.37).

2.1.76 object: An entity with a well-defined boundary and identity that encapsulates state and behavior. All objects are instances of classes (i.e., a concrete manifestation of a class is an object).

2.1.77 operation: A service that may be requested from any object of the class in order to effect behavior. Operations describe what a class is allowed to do and may be a request or a question. A request may change the state of the object but a question should not.

2.1.78 peer entities: Entities within the same layer.

2.1.79 pending command: From the point of view of the application client, the description of command between the time that the application client calls the **Send SCSI Command** SCSI transport protocol service and the time one of the SCSI target device responses described in 5.5 is received.

2.1.80 port: Synonymous with SCSI port (see 2.1.104).

2.1.81 power cycle: Power being removed from and later applied to a SCSI device.

2.1.82 power on: A condition resulting from a power on event (see 2.1.83).

2.1.83 power on event: Power being applied to a SCSI device, triggering a power on condition (see 2.1.82) in the SCSI device.

2.1.84 procedure: An operation that is invoked through an external calling interface.

2.1.85 procedure call: The model used by this standard for the interfaces involving both the SAL (see 2.1.96) and STPL (see 2.1.111), having the appearance of a programming language function call.

2.1.86 protocol: A specification and/or implementation of the requirements governing the content and exchange of information passed between distributed entities through the service delivery subsystem.

2.1.87 queue: The arrangement of tasks within a task set (see 2.1.139), usually according to the temporal order in which they were created.

2.1.88 receiver: A client or server that is the recipient of a service delivery transaction.

2.1.89 reference model: A standard model used to specify system requirements in an implementation-independent manner.

2.1.90 request: A transaction invoking a service.

2.1.91 request-response transaction: An interaction between a pair of distributed, cooperating entities, consisting of a request for service submitted to an entity followed by a response conveying the result.

2.1.92 request-confirmation transaction: An interaction between a pair of cooperating entities, consisting of a request for service submitted to an entity followed by a response from the entity confirming request completion.

2.1.93 reset event: A SCSI transport protocol specific event that triggers a hard reset (see 6.3.2).

2.1.94 response: A transaction conveying the result of a request.

2.1.95 role: When using class diagrams, a lable at the end of an association or aggregation that defines a relationship to the class on the other side of the assocation or aggregation.

2.1.96 SCSI application layer (SAL): The protocols and procedures that implement or issue SCSI commands and task management functions by using services provided by a SCSI transport protocol layer.

2.1.97 SCSI device: A device that contains one or more SCSI ports that are connected to a service delivery subsystem and supports a SCSI application protocol.

2.1.98 SCSI device identifier: Synonymous with SCSI port identifier (see 2.1.105).

2.1.99 SCSI device name: A name (see 2.1.73) of a SCSI device that is world wide unique within the SCSI transport protocol of a SCSI domain in which the SCSI device has SCSI ports (see 4.7.7). The SCSI device name may be made available to other SCSI devices or SCSI ports in SCSI transport protocol specific ways.

2.1.100 SCSI I/O system: An I/O system, consisting of two or more SCSI devices, a SCSI interconnect and a SCSI transport protocol that collectively interact to perform SCSI I/O operations.

2.1.101 SCSI identifier: A term used by previous versions of this standard (see Annex B).

2.1.102 SCSI initiator device: A SCSI device containing application clients and SCSI initiator ports that originates device service and task management requests to be processed by a SCSI target device and receives device service and task management responses from SCSI target devices. When used this term refers to SCSI initiator devices or SCSI target/initiator devices that are using the SCSI target/initiator port as a SCSI initiator port.

2.1.103 SCSI initiator port: A SCSI initiator device that acts as the connection between application clients and the service delivery subsystem through which requests and confirmations are routed. In all cases when this term is used it refers to an initiator port or a SCSI target/initiator port operating as a SCSI initiator port.

2.1.104 SCSI port: A class within a SCSI device that connects the application client, device server or task manager to the service delivery subsystem through which requests and responses are routed. SCSI port is synonymous with port. A SCSI port is either a SCSI initiator port (see 2.1.103) or a SCSI target port (see 2.1.108).

2.1.105 SCSI port identifier: A value by which a SCSI port is referenced within a domain. The SCSI port identifier is either an initiator port identifier (see 2.1.58) or a target port identifier (see 2.1.129).

2.1.106 SCSI port name: A name (see 2.1.73) of a SCSI port that is world wide unique within the SCSI transport protocol of the SCSI domain of that SCSI port (see 4.7.8). The name may be made available to other SCSI devices or SCSI ports in that SCSI domain in SCSI transport protocol specific ways.

2.1.107 SCSI target device: A SCSI device containing logical units and SCSI target ports that receives device service and task management requests for processing and sends device service and task management responses to SCSI initiator devices. When used this term refers to SCSI target devices or SCSI target/initiator devices that are using the SCSI target/initiator port as a SCSI target port.

2.1.108 SCSI target port: A SCSI target device that contains a task router and acts as the connection between device servers and task managers and the service delivery subsystem through which indications and responses are routed. When this term is used it refers to a SCSI target port or a SCSI target/initiator port operating as a SCSI target port.

2.1.109 SCSI target/initiator device: A SCSI device that has all the characteristics of a SCSI target device and a SCSI initiator device.

2.1.110 SCSI target/initiator port: A SCSI device that has all the characteristics of a SCSI target port and a SCSI initiator port.

2.1.111 SCSI transport protocol layer (STPL): The protocol and services used by a SCSI application layer to transport data representing a SCSI application protocol transaction.

2.1.112 SCSI transport protocol service confirmation: A procedure call from the STPL notifying the SAL that a SCSI transport protocol service request has completed.

2.1.113 SCSI transport protocol service indication: A procedure call from the STPL notifying the SAL that a SCSI transport protocol transaction has occurred.

2.1.114 SCSI transport protocol service request: A procedure call to the STPL to begin a SCSI transport protocol service transaction.

2.1.115 SCSI transport protocol service response: A procedure call to the STPL containing a reply from the SAL in response to a SCSI transport protocol service indication.

2.1.116 sender: A client or server that originates a service delivery transaction.

2.1.117 sense data: Data returned to an application client in the same I_T_L_Q nexus transaction (see 2.1.51) as a CHECK CONDITION status (see 5.9.6). Fields in the sense data are referenced by name in this standard. See SPC-3 for a complete sense data format definition. Sense data may also be retrieved using the REQUEST SENSE command (see SPC-3).

2.1.118 sense key: A field in the sense data (see 2.1.117 and SPC-2).

2.1.119 server: An entity that performs a service on behalf of a client. This standard defines one server, the device server.

2.1.120 service: Any operation or function performed by a SCSI object that is invoked by other SCSI objects.

2.1.121 service delivery failure: Any non-recoverable error causing the corruption or loss of one or more service delivery transactions while in transit.

2.1.122 service delivery subsystem: That part of a SCSI I/O system that transmits service requests to a logical unit or SCSI target device and returns logical unit or SCSI target device responses to a SCSI initiator device.

2.1.123 service delivery transaction: A request or response sent through the service delivery subsystem.

2.1.124 signal: (n) A detectable asynchronous event possibly accompanied by descriptive data and parameters. (v) The act of generating such an event.

2.1.125 standard INQUIRY data: Data returned to an application client as a result of an INQUIRY command. Fields in the standard INQUIRY data are referenced by name in this standard and SPC-2 contains a complete definition of the standard INQUIRY data format.

2.1.126 target: A term used by previous versions of this standard (see Annex B).

2.1.127 target device name: A SCSI device name (see 2.1.99) of a SCSI target device. See 4.7.2.

2.1.128 target identifier: A term used by previous versions of this standard (see Annex B).

2.1.129 target port identifier: A value by which a SCSI target port is referenced within a domain. See 4.7.2.

2.1.130 target port name: A SCSI port name of a SCSI target port or of a SCSI target/initiator port when operating as a SCSI target port (see 4.7.2).

2.1.131 target/initiator device name: A SCSI device name (see 2.1.99) of a SCSI target/initiator device. See 4.7.3.

2.1.132 task: A class within the logical unit representing the work associated with a command or a group of linked commands.

2.1.133 task tag: An attribute of a task containing up to 64 bits that is a component of an I_T_L_Q nexus. See 4.11.

2.1.134 task management function: A task manager service capable of being requested by an application client to affect the processing of one or more tasks.

2.1.135 task management request: A request submitted by an application client, invoking a task management function to be processed by a task manager.

2.1.136 task management response: The response returned to an application client by a task manager on completion of a task management request.

2.1.137 task manager: A server within a logical unit that controls the sequencing of one or more tasks and processes task management functions.

2.1.138 task router: A class within a SCSI target port that routes commands and task management functions between the service delivery subsystem (see 2.1.122) and the appropriate logical unit's task manager (see 2.1.137).

2.1.139 task set: A group of tasks within a logical unit, whose interaction is dependent on the task management (e.g., queuing) and ACA requirements. See 4.8.

2.1.140 third-party command: A SCSI command that requires a logical unit within a SCSI target device to assume the SCSI initiator device role and send SCSI command(s) to another SCSI target device.

2.1.141 transaction: A cooperative interaction between two entities, involving the exchange of information or the processing of some request by one entity on behalf of the other.

2.1.142 unconfirmed SCSI transport protocol service: A service available at the SCSI transport protocol service interface that does not result in a completion confirmation.

2.1.143 unlinked command: A SCSI command having the LINK bit set to zero in the CDB CONTROL byte.

2.1.144 well known logical unit: A logical unit that only performs specific functions. See 4.9.9. Well known logical units allow an application client to issue requests to receive and manage specific information usually relating to a SCSI target device.

2.1.145 well known logical unit number (W-LUN): The logical unit number that identifies a well known logical unit.

2.2 Acronyms

ACA	Auto Contingent Allegiance (see 2.1.8)
CDB	Command Descriptor Block (see 2.1.17)
LUN	Logical Unit Number (see 2.1.70)
MMC-2	SCSI Multi-Media Commands -2 (see 1.3)
n/a	Not Applicable
RAID	Redundant Array of Independent Disks
SAL	SCSI application layer (see 2.1.96)
SBC	SCSI-3 Block Commands (see 1.3)
SCSI	The architecture defined by the family of standards described in 1.3
SPI-5	SCSI Parallel Interface -5 (see 1.3)
SPC-2	SCSI Primary Commands -2 (see 1.3)
SPC-3	SCSI Primary Commands -3 (see 1.3)
STPL	SCSI transport protocol layer (see 2.1.111)
SSC	SCSI-3 Stream Commands (see 1.3)
VPD	Vital Product Data (see SPC-2)
W-LUN	Well known logical unit number (see 2.1.145)

2.3 Keywords

2.3.1 invalid: A keyword used to describe an illegal or unsupported bit, byte, word, field or code value. Receipt by a device server of an invalid bit, byte, word, field or code value shall be reported as error.

2.3.2 mandatory: A keyword indicating an item that is required to be implemented as defined in this standard.

2.3.3 may: A keyword that indicates flexibility of choice with no implied preference (synonymous with "may or may not").

2.3.4 may not: A keyword that indicates flexibility of choice with no implied preference (synonymous with "may or may not").

2.3.5 obsolete: A keyword indicating that an item was defined in prior SCSI standards but has been removed from this standard.

2.3.6 option, optional: Keywords that describe features that are not required to be implemented by this standard. However, if any optional feature defined by this standard is implemented, then it shall be implemented as defined in this standard.

2.3.7 SCSI transport protocol specific: Implementation of the referenced item is defined by a SCSI transport protocol standard (see 1.3).

2.3.8 reserved: A keyword referring to bits, bytes, words, fields, and code values that are set aside for future standardization. A reserved bit, byte, word, or field shall be set to zero, or in accordance with a future extension to this standard. Recipients are not required to check reserved bits, bytes, words, or fields for zero values. Receipt of reserved code values in defined fields shall be reported as error.

2.3.9 shall: A keyword indicating a mandatory requirement. Designers are required to implement all such mandatory requirements to ensure interoperability with other products that conform to this standard.

2.3.10 should: A keyword indicating flexibility of choice with a strongly preferred alternative; equivalent to the phrase "it is strongly recommended".

2.3.11 vendor specific: Specification of the referenced item is determined by the SCSI device vendor.

2.4 Editorial conventions

Certain words and terms used in this standard have a specific meaning beyond the normal English meaning. These words and terms are defined either in the glossary or in the text where they first appear.

Upper case is used when referring to the name of a numeric value defined in this specification or a formal attribute possessed by an entity. When necessary for clarity, names of classes, procedure calls, arguments or discrete states are capitalized or set in bold type. Names of fields are identified using small capital letters (e.g., NACA bit).

Names of procedure calls are identified by a name in bold type, such as **Execute Command** (see clause 5). Names of arguments are denoted by capitalizing each word in the name. For instance, Sense Data is the name of an argument in the **Execute Command** procedure call.

Quantities having a defined numeric value are identified by large capital letters. CHECK CONDITION, for example, refers to the numeric quantity defined in table 21 (see 5.3.1). Quantities having a discrete but unspecified value are identified using small capital letters. As an example, TASK COMPLETE, indicates a quantity returned by the **Execute Command** procedure call (see clause 5). Such quantities are associated with an event or indication whose observable behavior or value is specific to a given implementation standard.

Lists sequenced by letters (e.g., a-red, b-blue, c-green) show no priority relationship between the listed items. Numbered lists (e.g., 1-red, 2-blue, 3-green) show a priority ordering between the listed items.

If a conflict arises between text, tables, or figures, the order of precedence to resolve the conflicts is text; then tables; and finally figures. Not all tables or figures are fully described in the text. Tables show data format and values.

Notes do not constitute any requirements for implementors.

2.5 Numeric conventions

Digits 0 through 9 in the text of this standard that are not immediately followed by lower-case "b" or "h" are decimal values. Digits 0 and 1 immediately followed by lower case "b" are binary values. Digits 0 through 9 and the upper case letters "A" through "F" immediately followed by lower-case "h" are hexadecimal values.

Large numbers are separated by spaces (e.g., 12 345, not 12,345).

2.6 Notation conventions

2.6.1 Class diagram conventions

The notation in this subclause is based on the Unified Modeling Language (UML) specification.

Figure 1 shows the notation used for classes in class diagrams.

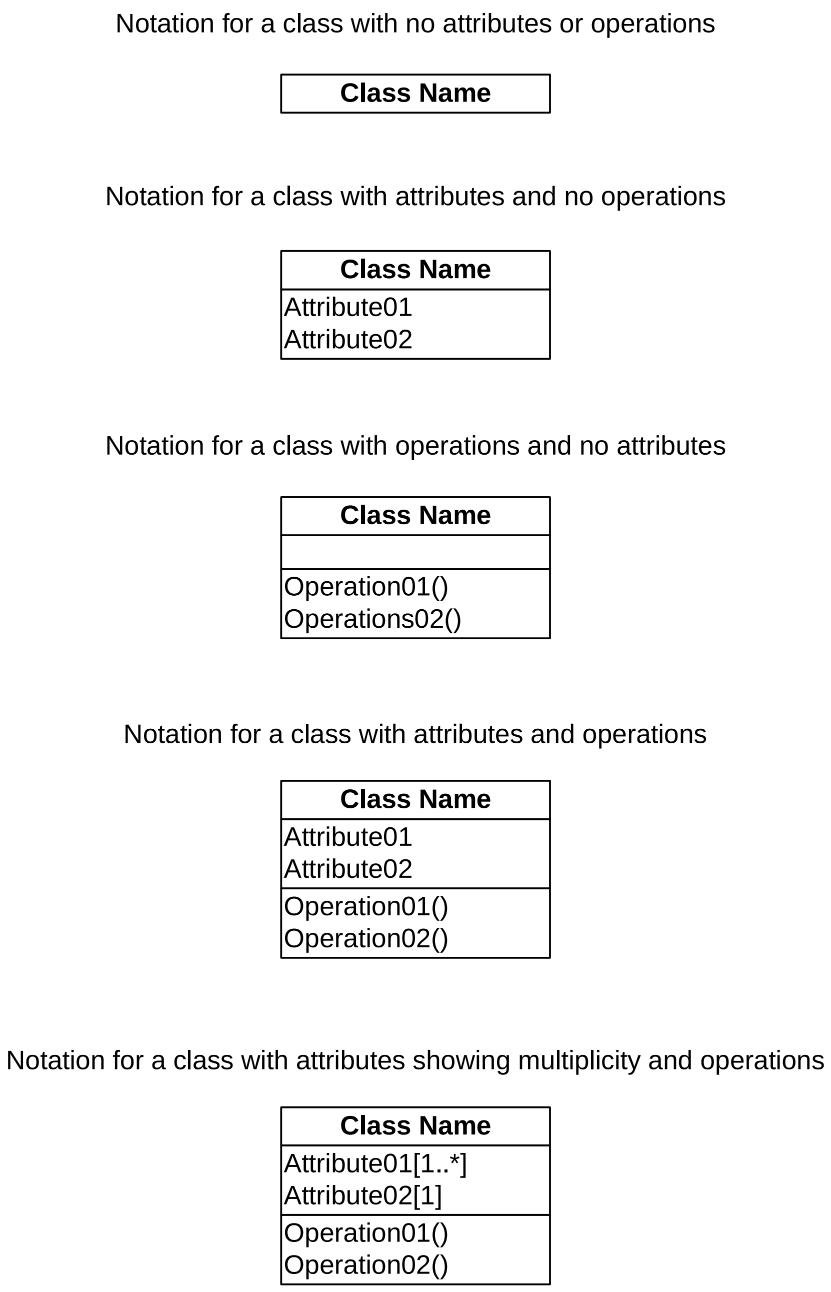

Notation for a class with no attributes or operations
Class Name
Notation for a class with attributes and no operations
Class Name
Attribute01
Attribute02
Notation for a class with operations and no attributes
Class Name
Operation01()
Operations02()
Notation for a class with attributes and operations
Class Name
Attribute01
Attribute02
Operation01()
Operation02()
Notation for a class with attributes showing multiplicity and operations
Class Name
Attribute01[1..*]
Attribute02[1]
Operation01()
Operation02()

Figure 1 — Class diagram conventions

See table 1 for the notation used to indicate multiplicity.

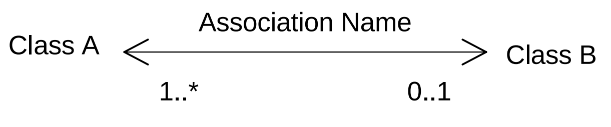
Table 1 — Multiplicity notation

Notation	Description
	The number of instances of an attribute is not specified.
1	One instance of the object or attribute exists.
0..*	Zero or more instances of the object or attribute exist.
1..*	One or more instances of the object or attribute exist.
0..1	Zero or one instance of the object or attribute exists.
n..m	n to m instances of the object or attribure exists (e.g., 2..8).
x, n..m	Multple disjont instances of the object or attribure exists (e.g., 2, 8..15).

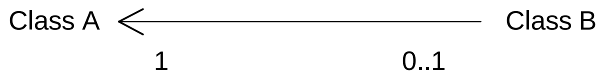
For the notation describing the relationships that may exist between classes see:

a) figure 2 for association;
b) figure 3 for aggregation;
c) figure 4 generalization; and
d) figure 5 dependency.

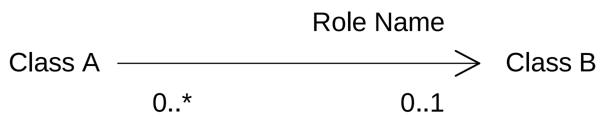
Association (“knows about” relationship)



Class A knows about Class B (i.e., is read as ‘Class A association name Class B’) and Class B knows about Class A (i.e., is read as ‘Class B association name Class A’)



Class B knows about Class A (i.e., is read as ‘Class B knows about Class A’) but Class A does not know about Class B



Class A knows about Class B (i.e., is read as ‘Class A uses the role name attribute of Class B’) but Class B does not know about Class A

Note: The role name and association name are optional

Examples of class diagrams using associations

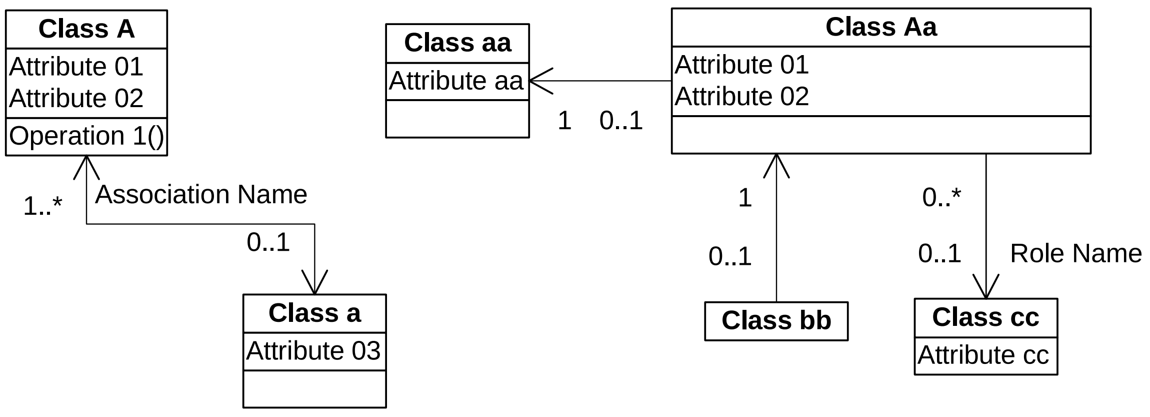
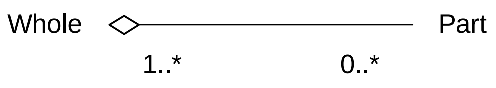
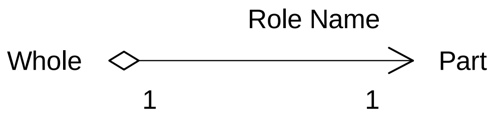


Figure 2 — Notation for association relationships for class diagrams

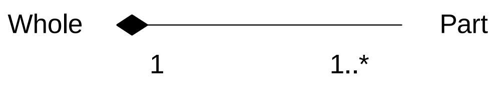
Aggregation (“is a part of” or “contains” relationship)



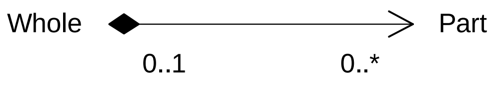
The part is part of the whole, knows about the whole, and may continue to exist even if the whole is removed.



The part is part of the whole, the part doesn’t know about the whole, and may continue to exist even if the whole is removed.



The part is part of the whole, knows about the whole, and shall not continue to exist if the whole is removed.



The part is part of the whole, the part doesn’t know about the whole, and shall not continue to exist if the whole is removed.

Note: The role name is optional

Examples of class diagrams using aggregation

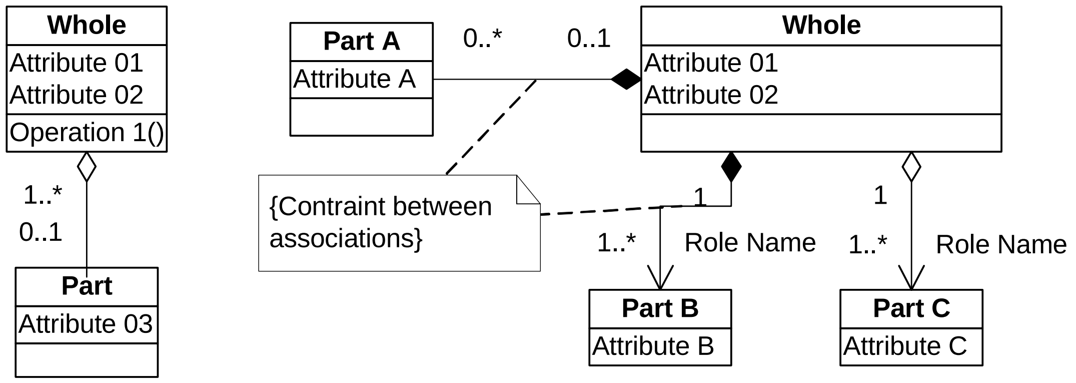
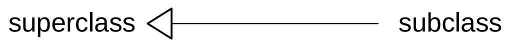


Figure 3 — Notation for aggregation relationships for class diagrams

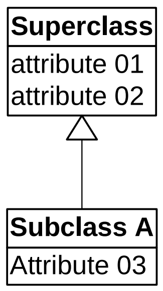
Generalization (“is a kind of” relationship)



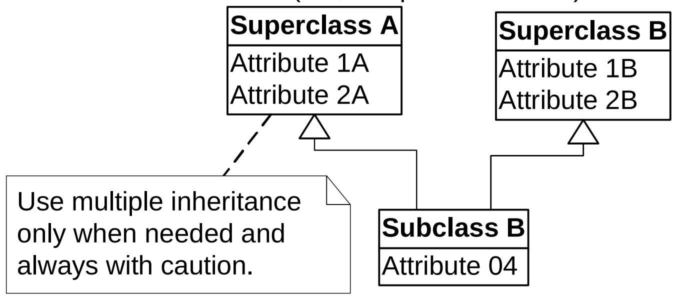
Subclass is a kind of superclass. A subclass shares all the attributes and operations of the superclass (i.e., the subclass inherits from the superclass).

Examples of class diagrams using generalization

Single superclass/single subclass:



Multiple superclass/single subclass
(i.e., muliple inheritance):



Single superclass/multiple subclass:

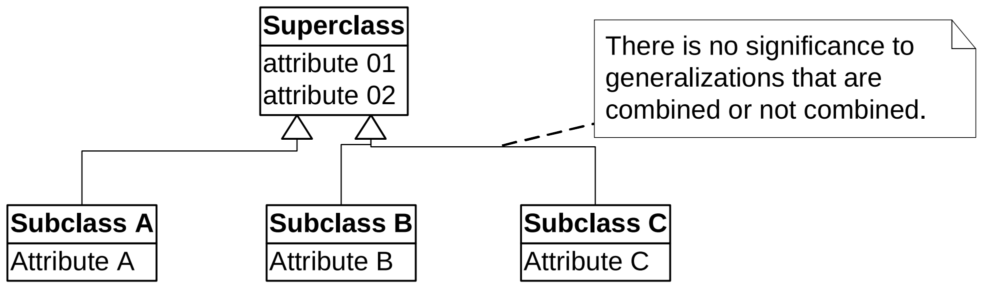
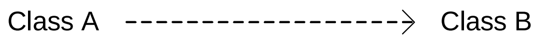


Figure 4 — Notation for generalization relationships for class diagrams

Dependency (“”depends on” relationship)



Class A depends on class B. A change in class B may cause a change in class A.

Example of class diagram using dependency

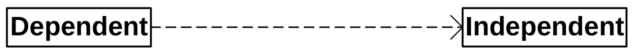


Figure 5 — Notation for dependency relationships for class diagrams

2.6.2 Notation for procedure calls

In this standard, the model for functional interfaces between entities is a procedure call (see 2.1.85). Such interfaces are specified using the following notation:

[Result =] Procedure Name (IN ([input-1] [,input-2] …]), OUT ([output-1] [,output-2] …))

Where:

Result:A single value representing the outcome of the procedure call.

Procedure Name:A descriptive name for the function modeled by the procedure call.

Input-1, Input-2, …:A comma-separated list of names identifying caller-supplied input arguments.

Output-1, Output-2, …:A comma-separated list of names identifying output arguments to be returned by the procedure call.

"[…]":Brackets enclosing optional or conditional arguments.

This notation allows arguments to be specified as inputs and outputs. The following is an example of a procedure call specification:

Found = **Search** (IN (Pattern, Item List), OUT ([Item Found]))

Where:

Found = Flag

Flag, if set to one, indicates that a matching item was located.

Input Arguments:

Pattern = … /* Definition of **Pattern** argument */

Argument containing the search pattern.

Item List = Item<NN> /* Definition of **Item List** as an array of NN **Item** arguments*/

Contains the items to be searched for a match.

Output Arguments:

Item Found = Item … /* Item located by the search procedure call */

This argument is only returned if the search succeeds.

2.6.3 Notation for state diagrams

All state diagrams use the notation shown in figure 6.

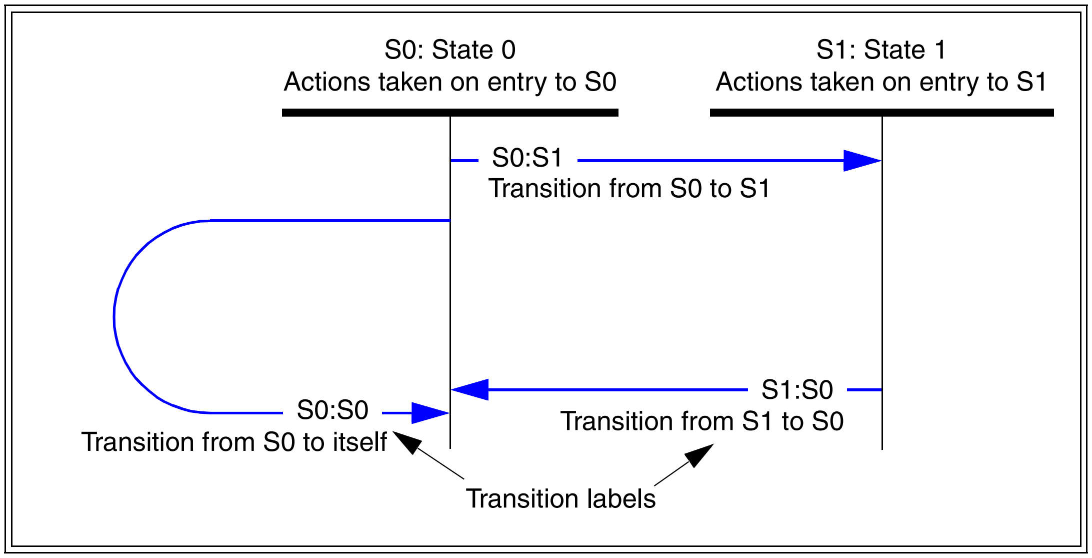


Figure 6 — Example state diagram

The state diagram is followed by a list of the state transitions, using the transition labels. Each transition is described in the list with particular attention to the conditions that cause the transition to occur and special conditions related to the transition. Using figure 6 as an example, the transition list might read as follows:

Transition S0:S1: This transition occurs when state S0 is exited and state S1 is entered.

Transition S1:S0: This transition occurs when state S1 is exited and state S0 is entered.

Transition S0:S0: This transition occurs when state S0 transitions to itself. The reason for a transition from S0 to itself is to specify that the actions taken whenever state S0 is entered are repeated every time the transition occurs.

A system specified in this manner has the following properties:

a) Time elapses only within discrete states;
b) State transitions are logically instantaneous; and
c) Every time a state is entered, the actions of that state are started. Note that this means that a transition that points back to the same state restarts the actions from the beginning.