LSC Language Reference

Preliminary Version

Assaf Marron

Department of Computer Science and Applied Mathematics

Weizmann Institute of Science

September 20, 2010

Table of Contents

Acknowledgements

The author would like to thank the members of David Harel's group at the Weizmann Institute for many discussions about this document, and for their valuable assistance, suggestions and comments. Besides David himself, they include Yoram Atir, Tal Berger, Michal Gordon, Amir Kantor, Shahar Maoz, Yaarit Natan, Avital Sadot, Itai Segall, Smadar Szekely and Guy Weiss.

This work was supported by the John von Neumann Minerva Center for the Development of Reactive Systems at the Weizmann Institute of Science, and by an Advanced Research Grant to David Harel from the European Research Council (ERC) under the European Community's FP7 Programme.

1. Basic Concepts

General

This document is a concise reference document for the language of LSC (Live Sequence Charts) based on the definitions in the original inception paper by Werner Damm and David Harel [**1**], and the books "Come, Let's Play" [**2**] and the Play Engine User Guide [**3**] by David Harel and Rami Marelly.

Scenario Based Programming

Designing and programming large systems is a complex and error prone task. Many languages, tools, and methodologies were developed over the years to make programmers more successful in producing software systems that meet their expectations. In his paper "Can Programming Be Liberated, Period?" [5] David Harel outlines a dream for liberating programming from the constraints which he summarizes as "the three straitjackets of programming":

1. The need to write down a program as a symbolic, textual, or graphical artifact;

2. The need to specify requirements (the what) separately from the program (the how) and to pit one against the other; and

3. The need to structure behavior according to the system's structure, providing each piece or object with its full behavior

The vision of liberated programming sees the programming process as something which is as natural and intuitive as instructing a person in a given assignment, or teaching a child new practical skills. In this dream world, the "programmer" / "teacher" / "author", would be focused on requirements and expectations and spend relatively less effort on implementation issues, such as architecture, objects, interfaces and traditional program code.

One approach towards liberating programming is Scenario Based Programming (SBP). In this paradigm, expectations and requirements from a computer system are specified in the form of use cases and typical sequences of events in the program execution.

By contrast to procedural and/or object oriented programming languages, such as C or Java™, in SBP, the building block of the resulting system is the scenario. The developer specifies multiple, possibly independent scenarios, each of which describes a sequence of user and system interactions which hopefully satisfy a certain purpose associated with a use case.

In a SBP environment, multiple scenarios can be combined automatically, to specify or generate a working system, which can execute those scenarios successfully.

SBP seems to be particularly appropriate for specifying reactive systems [6], but may be useful in many other contexts.

The LSC Language

The LSC language is based on adding modalities to Message sequence charts (MSCs), an ITU standard for a visual depiction of inter-object communication [7]. The UML [8] contains the concept of sequence diagram as well (influenced by MSCs, and by LSC).

The main distinguishing capabilities of the LSC programming environment include:

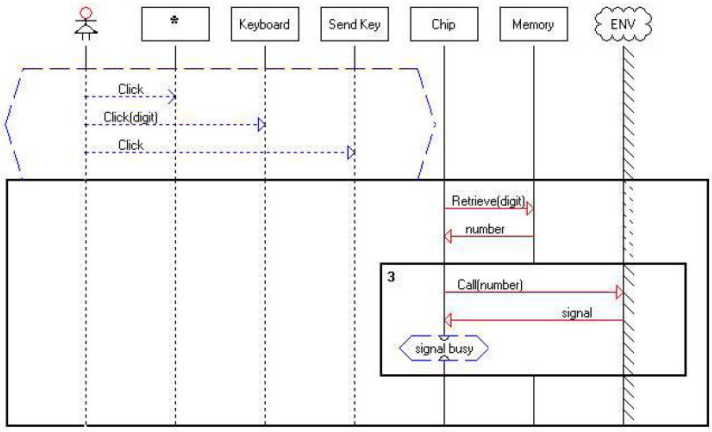
- Scenario Visualization: Depicting scenarios in easy-to-understand visual diagrams.
- Play In: In the LSC environment, in addition to menu driven drawing and specification capabilities, the user can enter events by actually causing them to happen in a GUI application that represents the external interfaces of the final system. Thus the developer can specify events by pressing buttons, turning on lights, entering text in display fields, or manipulating other objects in a mockup of the system being developed.
- Play Out: The LSC environment can execute the system as an integrative entity, using the collection of possibly autonomous scenarios.
- Multi-Modality: In LSC, execution elements are associated with multiple modalities, expressing, as do modal verbs in natural language, what must happen, what may happen, and what may not happen. The modalities supported by LSC are:
    - *must* vs. *may*: Using the terms hot and cold, one specifies which events and conditions must happen, or must be true at a given moment, and which are possible, but are not mandatory.
    - *allowed* vs. *forbidden*: In LSC there is a convenient and compact way to independently specify forbidden events or conditions, thus affecting the execution of multiple scenarios with little or no explicit inter-scenario communication.
    - *execution* vs. *monitoring*: In various contexts in LSC an expected event may be executed by the LSC environment (the Play Engine), or it may be executed by other means, and only be observed (or monitored) by the play-engine.
    - *existential* vs. *universal*: This modality distinguishes between constructs and requirements that apply to all possible runs of the system, and those that need to apply to at least one run of the system.

The LSC language is described in great detail in [**2**]. The book introduces the language gradually, and includes many examples, basic instructions for using the LSC development environment and the Play Engine, and explanations of reasoning behind various choices of the language designers. The book concludes with a complete formal specification of the language.

An LSC Example

Below is an LSC, which describes one of many scenarios in a telephone device. This LSC describes speed dialing, which is activated by pressing “*”, a digit key, and the “send” key. The

telephone then retrieves the number from memory, calls the number, and if the line is busy, attempts the call up to a total of three times.



The Play Engine

The development environment described in [**2**,**3**] is called “The Play Engine” (abbreviated PE).

The main capabilities of the play engine include:

- Play-in: Creating an LSC specification
- Play-out: Executing an LSC specification.

The PE is accompanied by an add-on, GUIEdit, which enables developing GUI applications that represent the final systems, and that are used during play-in and play-out.

Execution vs. Monitoring

Specifications (or portions thereof) in the LSC language can be interpreted in two distinct contexts or modes of operations:

- Execution: The specification drives the execution of the scenario. Subject to specified conditions and other controls, the events and interactions in the specification are actually performed by the PE (or in the developer’s head).

  This context can be used for creating the final system, or for simulating it (hence it is referred to also as *simulation*).

- Monitoring: In this context, also called *testing*, the LSC specification is used to validate that another running system, built with LSC or with other tools, operates as expected. The

sequence of actual events and interactions which takes place in the running system is checked against the monitored LSC specification, and any mismatches are indicated.

The LSC Specification

The LSC specification is a collection of individual live sequence charts. Each chart can be either existential or universal. An existential chart describes a possible scenario or sequence of events. A universal chart is divided into two parts – a prechart and a main chart. When the sequence of events in the prechart occurs[1] (as a result of being executed in other LSCs or in a monitored system), the chart is considered active and the sequence of events described in the main chart must be satisfied. Each universal LSC is marked whether it should drive execution or should be monitored. The PE attempts to execute the events specified in the LSCs marked for execution, and monitors the LSCs marked for monitoring – that is – it checks that events emanating from monitored system to match the specification in these charts. Existential charts are always monitored and do not drive execution.

The respective reference sections define in detail how such processes begin or end, and when they are considered successful.

Steps and Super Steps

Every execution of an event is considered a step. Sometimes, multiple events that are defined to be simultaneous are executed in a single step. In execution (simulation) context, some events are driven by the PE user, while others are triggered by the PE. Following a user action, the PE can optionally execute a *super step* – a sequence of steps which terminates when the next event must be a user action, or when the entire execution is terminated.

Strategies

In the Execution (simulation) context, the PE sometimes encounters a situation where multiple events are *enabled* - that is, there are several events which can occur next, as all their preceding events and other pre-requisite conditions were satisfied, per the LSC specification. The PE must then choose which event to execute (in Monitoring context, this is not an issue).

A strategy is then required to govern the selection of the next step. To date, play out can be governed by different strategies:

[1] Note that the term "occurrence of an event" is sometimes used in texts related to LSC for two distinct meanings: (a) the happening of the event at some point in time, and (b) the appearance of the event as part of the a specification in an LSC.

- Naïve Play Out: The next event to be executed is selected arbitrarily, from all enabled events.

- Smart Play Out: The PE uses model checking techniques to pick the next event from the set of enabled events. The PE challenges the model checker to prove that there is no super step which can take place without violating the LSC specification. If such a path actually exists, the model checker provides it as a counter example, and the PE can execute it. Alternate super-steps and execution paths can be found by dynamically adding to the model scenarios that forbid the sequences found so far.

- Planned Play Out: The PE uses an AI planning algorithm to create multiple alternate super steps, and also allows the end user to review those paths, select desired paths, and even back track.

- Maximal versus Minimal policies: Another factor in picking the next event for execution, from multiple candidate events, is whether the event is violating or not. In a minimal policy, violating events will be preferred over non violating events – minimizing the number of LSCs which may need to be processed (e.g. when cold violations cause precharts to be abandoned). In a maximal policy, non violating events are preferred, and violating events are deferred as long as possible, in an attempt to execute as much as possible of the specified behavior.

2. LSC Reference

General

This section summarizes concepts and constructs in the LSC visual language.

The order of entities in the reference is an attempt to enable linear reading. A concept that is needed for understanding the definition of other concepts usually precedes them. Forward references also exist, usually indicating further details, refinements and exceptions.

The terms in the reference are organized in groups:

- Basic Entities
- Basic Flow
- Additional LSC Concepts and Constructs
- Play Engine Terms

For each term in the reference, there may be several sections:

- Synonym Terms: Other names by which this entity may be known or referenced in LSC texts.
- Visual Notation: A textual description of the notation. Consolidated pictorial examples appear in the appendix.
- Semantics: Explanation of the meaning of the notation.

  When needed for highlighting specific differences for Execution or Monitoring – these appear in respective subsections.

  Note that the term *play out* is used both as the behavior of the PE (when using an LSC specification for execution or monitoring), and as a general term for the semantics of an LSC specification, regardless of any implementation.

- Notes and Variants: This section highlights specific issues and fine points

Basic Entities

System

Visual Notation

None

Semantics

The system is what is being developed or tested with the LSC specification.

Semantics for Execution

"The System" is an entity, comprised of an object system together with an LSC specification, with optional real-world interfaces thru a GUI application.

The system has behavior which is defined and driven by the LSC specification, and is manifested when the system is executed by a Play Engine. The behavior is determined by the LSC specification, the PE implementation choices, and play-out parameters chosen by the PE user.

Semantics for Monitoring

"The System" is a Monitored System - a device or an application (E.g. a telephone, or an airplane).

The behavior of the monitored system is dependent on its own implementation and on its environment, and is not dependent in any way on any Play Engine or any LSC specification (unless the monitored system is itself a PE execution of a collection of LSCs).

A Monitored System can be monitored with a Play Engine, using an LSC specification. When monitoring The System, the behavior of a PE (not of the system) depends on the PE implementation, the LSC specification of the expectations from The System, and the actual behavior of the [Monitored] System.

LSC specification

Visual Notation

None

Semantics

A collection of LSCs that together specify the [desired] behavior of the system

LSC

Synonym Terms

Live Sequence Chart; Scenario

Visual Notation

See visuals for Universal and Existential LSCs (See figure 4.1)

Semantics

A formal specification of system behavior in one scenario. There are two types of LSCs: Universal and Existential.

The term scenario may be used both as a synonym for LSC, and for a more abstract or conceptual sequence of events, that may be encoded by LSCs.

Notes and Variants

An LSC is associated with a name that distinguishes it from other LSCs.

Universal LSC

Visual Notation

A diagram characterized by a dashed hexagon, situated on top of a solid rectangle. (See Fig. 4.1)

Semantics

Semantics for Execution

A mandatory scenario. Whenever the events (or conditions) in the prechart section of the LSC (the hexagon) occur (or are satisfied), the behavior described in the main chart section of the LSC (the rectangle) begins to execute (and must be satisfied).

Semantics for Monitoring

A mandatory scenario. Whenever the events (or conditions) in the prechart occur (or are satisfied) in The [Monitored] System, the execution of The [Monitored] System is expected to proceed according to the specification in the main chart.

Notes and Variants

In Monitoring - the System may behave differently from what is specified by the LSC specification. In this case the monitoring will indicate a violation.

See also Life Cycle of LSCs.

Existential LSC

Visual Notation

A diagram characterized by having a dashed rectangle as its outermost frame. (See Fig. 4.1)

Semantics

A possible scenario. may or may not occur in a particular run. May be used as a test case or an example of an acceptable sequence of events. (See Consistency of a system with an LSC). Both in monitoring and in execution, existential charts are monitored, and their successful completion can be a useful indicator to the PE user.

Semantics for Execution

While existential charts may be part of the definition of the behavior of a system, they do not drive execution by the PE. They can influence execution by participating in consistency checking during synthesis of an object system from an LSC (a topic which is out of the current scope of this reference document).

Semantics for Monitoring

See Semantics paragraph above

Notes and Variants

See also Life Cycle of LSCs. All elements in an existential LSC must be cold.

Prechart

Visual Notation

A dashed hexagon at the top of a chart, sitting on top of a solid rectangle. (See Fig. 4.1)

Semantics

Semantics for Execution

The prechart describes events (and conditions), that when they occur (or are satisfied), The System will attempt to execute the specification in the main chart represented in the solid rectangle below the prechart.

The entry of the prechart and the exit of the prechart are synchronization points for the participating instances.

Semantics for Monitoring

The prechart specifies events (and conditions), that when they occur (or are satisfied), The Monitored System is expected to ("must") execute steps that correspond to the specification in the main chart.

Notes and Variants

All elements in a prechart must be cold.

Main Chart

Visual Notation

An outermost solid rectangle in a universal chart, situated below a prechart. (See Fig. 4.1)

Semantics

Semantics for Execution

The definition of a prechart for execution explains the semantics of a main chart.

Semantics for Monitoring

The definition of a prechart for monitoring explains the semantics of a main chart.

Chart

Notes and Variants

The term "chart" without qualification is used in this document to refer to either:
an existential LSC, the prechart or the main chart of a Universal LSC, a subchart, both subcharts in an "if-then-else" construct, and a loop construct.

Object

Visual Notation

See visualization for Instance.

Semantics

An object is an entity that can participate in scenario by sending events, receiving events, or by being subjected to time synchronization. It can have properties and methods. Each appearance of a given object in an LSC is called an instance. Events described in scenarios are represented as messages sent between objects.

Semantics for Execution

The object exists in the LSC specification.

Semantics for Monitoring

The term *object* is used both for objects in the LSC specification and concrete objects the monitored system.

Object Property

Synonym Terms

State Variable; System State Variable

Visual Notation

The forms of the PE show property values as text; GUI application may simulate properties visually - such as light being on or off, a display area having a background color, or a display area showing a specific value. Messages changing object properties are labeled with the name of the property and its value (see fig. 4.3).

Semantics

Each object may be associated with a set of properties which represent different state variables of the object. Property values can be changed as a result of messages received by the object. The semantics of each property depends on the application. Each property has property attributes which are part of LSC specification.

Properties are associated with property types, such as Boolean or String.

Method

Visual Notation

In LSC methods appear as part of object definitions. Method calls appear as labels of method call events. These labels include the method name followed by parameters in parentheses. (See Fig. 4.2)

Semantics

Methods are means for transferring data or control signals between objects. Each object (class) can have an arbitrary number of methods, which other objects can call in order to transfer the required information. Every method has a name and a set of parameters defined over the supported types. A method can be either synchronous or asynchronous, thus specifying whether the calling and the called objects synchronize when the method is called.

Notes and Variants

The LSC specification of each method doesn't describe what the method is intended to do and does not cause the execution of some method-specific code outside of the LSC specification. The method call event influences system operation by influencing the flow of LSCs, or by serving as anchors for integration with other systems.

Instance

Visual Notation

A vertical line with a solid rectangular label (header) with an object name on top. The line can be in parts solid and in parts dashed (see Fig. 4.1).

Semantics

Semantics for Execution

An instance is a representation (appearance) of an object of the executed system in one LSC.

Semantics for Monitoring

A representation of an object of the monitored system in one LSC.

Notes and Variants

By definition, instances for a given concrete object can appear in different LSCs.
The line of an instance is also called its "life-line".
The instance line is solid or dashed, based on the temperature (hot or cold) of its locations.
The instance line of the object Env (environment) is comb-like.
See also Symbolic Instance

Event

Visual Notation

An event is indicated visually by one or more locations and is associated with messages and other constructs. See individual constructs for visualization of different types of events (See fig. 4.1).

Semantics

An event is the basic component of a scenario. Examples of events or types of events include sending or receiving of a message , entering a chart, reaching the end of a chart, executing a primitive LSC construct (e.g., assignment, evaluating a condition, a clock tick, , or binding of a symbolic instance), or entering or exiting a compound LSC construct (e.g., a subchart, a loop, or if-then-else),.

An event can be hidden or visible. See execution rules for each type of event, and definition of minimal event.

An event is bound, if all the variables that appear in it are bound.

Semantics for Execution

During execution, an event can be triggered by a user action, or by the PE advancing instances thru locations. The occurrence of events is the execution of the system.

Semantics for Monitoring

During monitoring, visible events are generated by the monitored system, and hidden events are generated by the PE.

Message

Visual Notation

An Arrow from a sending object to a receiving object, labeled with either a desired change to a property or a method call.

Hot Message: The arrow line is solid and red
Cold Message: The arrow line is dashed and blue.

Synchronous message: The arrow head is a closed triangle
Asynchronous message: The arrow head is an open sharp angle. ">"

(See Fig. 4.2)

Semantics

A message is a type of event - representing an interaction. A message contains either:

- the name of a property of the receiving object, and a new value, or
- a method call (method name and parameters)

The interaction can be between two instances, between an instance and itself, an instance and the environment, or a user and an instance.

A message may be hot or cold.

- A hot message must be received after it is sent - but only in terms of communication line requirements;
- A cold message might not be received after it is sent.

A message may be synchronous or asynchronous - See details below.

If the sending instance is the same as the receiving instance, the message is called a "self message". The sending location is also the receiving location.

A message may be constant or symbolic

- A constant message: A message that contains no symbolic parameters.
- A symbolic message: A message where the value of the property, or parameters to a method call, are specified as a name of a variable
  - A symbolic message is bound if all its variables are bound. Otherwise it is free.

Semantics for Execution

Execution rules - Sending a Message:

- If the message is symbolic – binding will occur prior to execution.
- The sending instance proceeds past the sending location.
- For self message – the instance proceeds past the sending-and-receiving location.

Execution rules - Receiving a Message:

- If the message is an exact or symbolic property change (whether explicit or thru prefix) – the PE changes the property of the receiving instance. Otherwise (the message is a method call) – if some integration is specified into a GUI application – the PE invokes the integration, otherwise – the PE does nothing. (In "pure" LSC, the

PE does not implement the message, and its effect is in the changes it induces over the flow of control, the binding of variables and the behavior of other constructs).

- The current location of the instance becomes the one just after the receiving location.
- If the message is synchronous, the sending event and the receiving event are simultaneous.
- If the message is asynchronous, the receiving event is separate from, and occurs after, the sending event.

Semantics for Monitoring

- The “real meaning” of the message is whatever meaning it carries in the monitored system.
- The current location of the instance becomes the one just after the receiving location.
- The PE does nothing with the message, whether it is a method call or a property change.
- The concept of synchronous messages does not apply for monitoring. Events for sending and receiving of messages are treated separately. The PE does not validate that the sending and receiving of a synchronous message were processed simultaneously in the monitored system. The implementations may also treat them as one event, or may handle only the sending event or only the receiving event.

Notes and Variants

- Activities such as “clicking a button” or “turning on a switch” are represented as messages from a user instance to an object instance.
- For convenience and clarity, the name of the property can be replaced by a verb that reflects the changing of the property.
- The temperature of a message is a separate concept from the temperature of the sending and receiving location.
- In the standard semantics, the temperature of the receiving location overrides the temperature of the message.
- In an alternate semantics it is possible to implement the PE such that the temperature of a message overrides the temperature of the receiving location.
- The new value of the property that appears in the message can be constant or symbolic.
- A message is synchronous if and only if:
  - The message is a property change, and the property is defined as synchronous, or
  - The message is a method call, and the method is defined as synchronous method, or
  - The message is either a property change or a method call and is a self message.

Hidden Event

Visual Notation

See visuals for specific constructs.

Semantics

Hidden events include:

- entering or leaving a chart
- an assignment
- a condition
- entering, leaving or skipping a loop
- binding a symbolic instance with a concrete object
- evaluating forbidden condition

These events are internal to the LSC, and do not involve exchange of messages.
See detailed semantics for each type of event.

Visible Event

Synonym Terms

System Events

Visual Notation

See visual for specific constructs

Semantics

Visible events are a subset of all possible events (LSC events):

Definition 1: Messages (including method calls and clock ticks).
Definition 2: Those events that can possibly be observed in a monitored system, if the LSC is applied in monitoring
Definition 3: Events that are not hidden (per above list).

See execution rules for each type of event.

Notes and Variants

The terms visible event and system event are used also in the context of execution.

Basic Flow

Location

Visual Notation

The intersection of a lifeline of an instance with a message or another LSC entity. Optionally marked also with a solid filled circle. The color of the location reflects its temperature .

Hot locations: red filled circles followed by a solid line to the next location.

Cold locations: blue filled circles followed by a dashed line to the next location

(See fig. 4.1)

Semantics

Locations represent the state of an instance and mark the smallest unit of progression in the life of an instance. Each instance is associated with one initial location, which is marked at the label/header of the lifeline. Each of the other locations is associated with the occurrence of one event. Transitions move the current location of an instance from one location to another location.

A location may be hot or cold.

See definition for temperature of a location.

Notes and Variants

Locations apply only to the LSC. The monitored system is not aware in any way of locations.

The temperature of a location is set by the user.

The temperature of a location influences the temperature of a cut.

Current Location of an instance

Visual Notation

The intersection between a cut and an instance visually identifies the current location of that instance. (See Fig. 4.10)

Semantics

When an LSC becomes active, the instances which caused the activation of an LSC are at the location of their first event. All other instances are in their initial locations.

According to the rules of Transitions , each instance progresses thru its possible locations – changing the current location.

Temperature of an entity – Being Hot or Cold

Visual Notation

See visualization for the individual entities. In general the color red and solid lines indicate hotness, and the color blue and dashed lines indicate coldness. (See Fig 4.1)

Semantics

Certain LSC entities are assigned a temperature – being hot or cold. These are locations, messages, conditions, forbidden messages and forbidden conditions.

The temperature of entities is a factor in controlling the PE progression thru the locations and lifelines, and its assessment of how to treat events arriving from a monitored system

Generally hotness of a location implies that something must happen. Coldness indicates that nothing *has* to happen and some things may happen. In general, for a cold location, the instance may remain in this location indefinitely, and exiting the current chart from this location does not create a violation. For a hot location, in general, the instance must eventually proceed past the location, cannot stay in the location until system termination, and cannot exit the chart from that location. See Transition rules for specific constructs.

Semantics for Execution

The (only) ways to leave a hot location are either executing the event, or the occurrence of a cold forbidden event or a cold forbidden condition.

For a cold location – all transitions apply.

Semantics for Monitoring

When an instance is in a hot location:

- For visible events: the (only) ways to leave a hot location are either observing the event, or the occurrence of a cold forbidden event or a cold forbidden condition.

  For a cold location – all transitions apply.

- For hidden events - see semantics for execution.

Notes and Variants

For the most part, the temperature of each construct in each LSC is assigned by the author of the LSC.

For certain locations - temperatures are fixed and cannot be changed:

- Initial locations are hot
- All prechart locations (and entry into a main chart) are cold
- All existential chart locations are cold

Cut

Visual notation

A hatched (comb-like) step line, either red or blue, thick and solid that crosses once each instances in an LSC. (See Fig 4.10)

Semantics

A cut exists during a run. A cut is a mapping of each instance in a live copy of an LSC to its current location. The location which is immediately below the intersection of the cut with each instance is that instance's current location.

A cut may be hot or cold

- A hot cut is a cut where at least one of the instances is in a hot location.
- A cold cut: A cut that is not hot.

See temperature for a general description of the difference in behavior between hot cuts and cold cuts. See transitions for more details.

Notes and Variants

During play-out the PE displays the live copies of LSCs, each with its cut.

The formal definition of a cut involves an interim term of *Legal Cut* - which ties the definition to the orchestrated transition of locations in all LSCs.

An alternate semantics is possible, where a cut is hot also if all locations are cold but one of the locations is associated with a HOT message.

Live Copy of an LSC

Visual Notation

See cut

Semantics

During play out, the PE matches (unifies) events to the minimal events in existential LSCs and precharts of universal LSCs. When such matching is successful, the PE creates a live copy of the LSC, and from that point on, this live copy participates in dictating the progress of the play-out.

The live copy contains a copy of the LSC, current activation mode, and current cut. Its modes can be preactive or active. (The LSC is said to be "running").

Configuration

Synonym Terms

State

Visual Notation

See visualization for cut.

Semantics

A configuration of an LSC specification is the collection of all live copies of LSCs , and an indicator whether this configuration is violating or not.

Run

Synonym Terms

A trace; A system run; Executing a system

Visual Notation

(a) A collection or sequence of events (as in a trace file or log file)

(b) Multiple configurations, displayed as a sequence of steps.

Semantics

Semantics for Execution

A run of the system, or executing a system, means executing an LSC specification. The user applies one or more external stimuli. The steps in the LSC are then executed according to transitions.

The run is sometimes equated with the sequence of events generated by the execution.

An execution run begins with an LSC specification and an empty set of live copies of LSCs. An event (such as PE user action, or a clock tick) may trigger at least one LSC in the specification – that is, matches a minimal event in an existential chart or in a prechart of a universal chart, and thus causes the creation of a live copy of the LSC.

An execution run ends either when a violation occurs, or when all live copies of LSCs have been completed.

Semantics for Monitoring

Monitoring a system means activating the monitored system in its native (LSC or non LSC) environment, and checking whether its behavior is in accordance with the LSC specification. Part of the monitoring includes performing required transitions in the LSC, thus updating the current configuration. The events in the monitored system are not violating only if they are enabled in the then-current LSC configuration.

The run is sometimes equated with the sequence of events generated by the execution.

A monitoring run begins with an LSC specification and an empty set of live copies of LSCs. Event that occurs in the monitored system and appears in the specification are processed by the PE. A monitoring run ends, when a violation occurs, or when the PE user determines that the last event from the monitored system was processed.

Notes and Variants

Executing an LSC specification includes some monitoring aspects, in that events generated by one LSC in the executed LSC specification are unified with events in other LSCs, in much the same way as this is done for a monitored system. Thus LSCs in the specification are triggered/activated, advanced, satisfied or violated, based on events generated by the execution.

Execution tools such as the Play Engine provide a host of additional visual information beyond the visuals provided here.

A run may also be associated with a trace of events that happened during the run.

Enabled event

Visual Notation

Red, hollow circle - thin or thick, immediately below a cut line.

An appearance of an enabled event in an LSC, which is selected to be executed next by the PE, is depicted as a thick circle. Additional appearances of the same event which will be advanced simultaneously with the selected occurrence are depicted as thin red circles.

(See Fig. 4.10)

Semantics

Being enabled is a state of an event with respect to a given cut (in a given LSC), during a run.

An event is enabled only if all the events that should have occurred prior to this execution occurrence specification have already happened.

For a pair of events which are the sending and receiving side of the same synchronous message, the above condition must hold for both the sending location and the receiving location.

An event of receiving an asynchronous message is enabled when the above condition holds, and the message has been already sent (the sending instance advanced past the sending location).

An event of executing an assignment requires additionally that all variables that appear in its right hand side, be bound.

A hot event of evaluating a condition is enabled if all its constituent conditions are evaluated to TRUE.

Notes and Variants

Multiple events can be enabled at a given time.

Execution of an event

Visual Notation

See visual under Run.
Shown by progression of the Play Engine.

Semantics

Semantics for Execution

Processing an event specified in the LSC, subject to the Semantics for Execution and the rules of Transition.

Semantics for Monitoring

Processing an event that came from the Monitored System, subject to the Semantics for Monitoring and the rules of Transition.

Notes and Variants

- Only enabled events can occur or be processed successfully without violation.
- An event that is enabled in one LSC may cause a violation in another LSC.
- When monitoring a system one does not know whether the event that occurred changed the properties of the objects of the monitored system. This will be known only thru monitoring their behavior in future events.

Violation of an LSC by an event

Visual Notation

Cold violation: A blue diagonal cross over a red rectangle surrounding the entire LSC. (See Fig. 4.11)

Hot violation: A red diagonal cross over a red rectangle surrounding the entire LSC.

Semantics

In general, an event violates an LSC (hot or cold violation), if the chart is strict, the event appears in the chart, and the event is not enabled in the chart.

Cold violation: (a) Violating a prechart (or an existential chart): Once an event occurs, it violates a preactive prechart of an LSC (or an existential chart), if it is unified with an event that appears in that prechart (or existential chart) but is not enabled there, per the current cut, or it is unified with a cold forbidden event in that chart, or it creates a condition which is cold forbidden. The prechart (or existential chart) is then abandoned.
(b) Cold violation of a main chart: Once an event occurs, it cold-violates the main chart of an LSC if it is unified with an event that appears in that main chart but is not enabled there by the current cut, and the current cut is cold, or it is unified with a cold forbidden events in that chart, or it creates a condition which is cold forbidden. The main chart is then abandoned.

A cold violation is not considered inconsistent with the specification and does not have other effects on the run.

Violation (= Hot violation): Once an event occurred it may violate a main chart of a universal LSC if either :

- the event appears in that LSC but is not enabled there, per the current cut, and the LSC is strict, and the LSC is active and the current cut is hot, or,
- the event is unified with a hot forbidden event in that chart (regardless of whether the current cut is hot or cold), or
- the event creates a condition which is hot forbidden (regardless of whether the current cut is hot or cold).

Once a violation (=hot violation) occurs during a run, the run stops. Play-out tries to avoid hot violations when possible.

Notes and Variants

The term hot violation and cold violations are informal. The term cold violation should probably be replaced, by a less confusing term.

Completion of a prechart

Visual Notation

A cut with all objects at the beginning of the main chart.

Semantics

All instances in the LSC are synchronized at their intersection with the main chart.

Completion of a main chart

Visual Notation

Solid blue frame around the chart.

Semantics

The current location of all instances is at the end (bottom line) of the main chart. The execution completed successfully for this chart. The chart was satisfied. The live copy of the chart is discarded.

Satisfaction of an existential LSC by a run

Visual Notation

See satisfaction of an LSC specification

Semantics

A run satisfies an existential LSC if it caused the existential LSC to be in check mode (a minimal visible event in the chart occurred), and then the chart eventually completed.

Satisfaction of an LSC specification by a given run

Synonym Terms

Consistency of a run with an LSC specification

Visual Notation

Visual depiction is same as system run above - at a final stage

Semantics

Semantics for Execution

A run that ends without a violation.

Semantics for Monitoring

An execution of the monitored system which is in full agreement with the LSC specification. No violation occurred: All the dictated events occurred. No forbidden events occurred. Specified events occurred only when expected.
The set of live copies of LSCs is empty .

Notes and Variants

A run of a monitored system may be empty, the actual events that occurred in the monitored system may not activate any LSCs.

Consistency of a System with an LSC specification

Visual Notation

None

Semantics

We say that a system is consistent with an LSC specification if all its runs satisfy the LSC specification, and for each existential LSC in the specification, there exist at least one run of the system that satisfies it.

Additional LSC Concepts and Constructs

Variable

Visual Notation

The name of the variable, such as "X5", or "Altitude" appears as a string, in messages, conditions, and other constructs. (See Fig. 4.8)

If a variable is bound, the value of the variable appears as a tooltip next to the variable name upon mouse-over.

Semantics

A symbol that may appear in constructs, and which may assume different values at different times according to execution rules and unification processes.

Scope: A variable is local to an LSC (a chart), and therefore can be referred to only within the chart in which it is defined. If multiple live copies of a given LSC are active, each has its own set of variables. All occurrences of a given variable name in a given activation of a given chart refer to one variable.

A variable may be bound or free.

- We say that a variable is bound if and only if, the symbolic name is associated with a value. Otherwise we say that the variable is free

Each variable is specified with a type, which controls the possible values of the variable.

The initial state of a variable in an LSC is free, and it is bound as a result of either:
- Execution of an assignment statement
- Unification

Connector

Visual Notation

A closed semi-circle on a condition or assignment at their intersection with an instance line. (See Fig. 4.8)

Semantics

All instances whose intersections with a given construct (condition or an assignment) are marked with a connector, are synchronized before the construct is processed.

Assignment

A rectangle with a folded top-right corner ("dog-ear"), containing a variable name on the left, an assignment symbol ":=" in the middle, and an expression on the right hand side.

Semantics

An assignment stores a new value in a variable. The new value may be:

- A constant
- The value of an object property
- The value of an evaluated implemented function
- The current Time

The entrance to an assignment is a synchronization point.
Participating instances are marked by connectors at the intersection point.

An assignment is executed as soon as it is possible after the previous event was executed (and we say that the assignment is then enabled) – that is:

- when all instances that are synchronized with it have reached it, and
- when all the variables in the right hand side are bound.

Notes and Variants

An alternative semantics could have been implemented, where the execution of an assignment is delayed until just before the next visible event which follows it, instead of the above.

Condition

Synonym Terms

Assertion; Evaluation of a condition; Constraining of a variable

Visual Notation

Hot condition: red solid hexagon containing a condition expression (see below) (See Fig. 4.8)

Cold condition: blue dashed hexagon containing a condition expression (see below) (See Fig 4.5)

Timing constraint: a hot or cold condition with an hour-glass (two triangles) at the top right hand of the condition (See Fig 4.8)

Semantics

A requirement or limitation – expressed as one of the following:

- a relation, such as equal, not equal, greater than, etc. between a property value or a variable value, and a constant value or another variable or a function call. It evaluates in the natural way.
- a free expression:
  - TRUE or SYNC: always evaluate to true
  - FALSE: always evaluates to false.

- a SELECT expression with a pair of numbers that add up to 100%: Evaluates to true with the probability indicated by the first (leftmost) number.
- A timing constraint: The word TIME (which represent the current time) and a relation to an expression that includes a variable (which contains some time value). Evaluates to true or false in the natural way.
    - A minimal delay is a condition of the form Time > TimeVariable + MinDelay
    - A maximal delay is a condition of the form Time < TimeVariable + MaxDelay
- a conjunction of the above.

A condition may be hot or cold.

The entry into a condition is a synchronization point.

Execution Rules - Hot condition:

- Connected objects are synchronized.
- If the condition contains the constant free condition “FALSE” the main chart is violated.
- If the condition contains a false timing constraint that cannot become true by only waiting and allowing time to pass (the time has already passed) – the main chart is violated.
- The condition is evaluated constantly and repeatedly, waiting for the condition to become true. (until then this instance is "hung" and cannot proceed.); When true - The next location in each synchronized instance is enabled.

Execution Rules – Cold condition

- Connected objects are synchronized
- The condition is evaluated once.
    - If the condition is true - all synchronized instances proceed in the current chart.
    - If the condition is false - the current chart is exited.

Semantics for Execution

See above.

Semantics for Monitoring

SELECT expressions are not evaluated. Both possibilities are allowed.

Notes and Variants

- A condition which is minimal in a prechart is evaluated once the visible event below it occurs.

- One specific timing constraint is Hot Message Delay. This is a form of specifying minimum and/or maximum time delays for the receipt of a message, and is accomplished by using other constructs.

  The sending time is stored in the sending instance – the assignment is placed immediately after the sending event (which causes the assignment to be executed at the same time as the sending of the event). A hot minimal delay is placed before the receipt of the message, forcing synchronization (to bind the variable holding the time value). A hot maximal delay can be placed after the receipt of the message.

  The receiving instance is forced to WAIT for the minimal delay duration before receiving the message. If more time has passed than the maximal delay – the LSC is violated.

If-then-else

Visual Notation

Two subcharts and a condition hexagon inside (at the top) of the first one (See Fig. 4.4)

The second subchart is optional. (See Fig 4.6).

Semantics

A branching construct. Comprised of an IF-part subchart, that is a subchart that starts with a condition, and an optional second subchart - the ELSE-part subchart;

The entrance and exit of each of the subcharts is a synchronization point.

For main chart:
The construct is evaluated as soon as possible (that is, when all participating instances are synchronized with it, and all involved variables are bound.

- Participating objects are synchronized
- The condition is evaluated once
- If the condition is true, the next event in the IF-part is enabled. Subsequent events proceed as usual. When the IF-part is exited, the ELSE-part is skipped.
- If the condition is false, the IF-part is skipped, the ELSE-part subchart is entered. Subsequent events proceed as usual.

For prechart and existential chart: The chart is monitored until either the IF condition is TRUE and the first visible event in the IF-part occurs, or until the IF condition is not true and the first event in the ELSE-part occurs. Subsequent events proceed as usual.

Loop

Visual Notation

Subchart with loop control mark on left top corner (*, ?, <n>, or <variable name>). (See Fig. 4.5)

Semantics

An LSC construct that controls iterative execution. It is comprised of the loop control and the loop body.
Execution Rules:
- Participating objects are synchronized.
- if the loop control is a numeric constant or a variable, its value is saved.
- if the loop control is "?" - the user is prompted to enter a value for the number of iterations.
- if the loop control is “*”, the loop is unbounded, that is, the loop control is assumed to be an infinite number.
- the loop body is executed (according to other execution rules).
- the loop is exited when either a false condition forces an exit from the current subchart, or, the saved, finite, number of iterations is exhausted.
- the handling of variable scope and initialization within a loop is described in the transition procedure.

Clock

Visual Notation

Lifeline: An instance with clock icon instead of a label. (See Fig. 4.9),
GUI Object: A rectangle with a numeric value (00.000 at initialization) on the PE toolbar, next to the Tick button.

Semantics

The clock is an object with a single property Time, and a single method Tick.

One clock object exists in every system, and an instance automatically and implicitly exists in every LSC, even if it is not drawn.

The clock instance is drawn in an LSC, by the PE, when playing-in a first Tick event.

The property Time is assigned an arbitrary value at the beginning of execution of an LSC specification.

The value of this property is incremented in quanta according to the passage of time during play-out.

The Tick event is executed during play out according to the passage of time, when the Time property is incremented.

The property is available for conditions and assignments even if the clock object is not drawn in the chart.

Tick events can be only positively unified with an event. They cannot be negatively unified and cannot directly cause a violation through unification. Tick events can be used as forbidden elements

Semantics for Execution

See above

Semantics for Monitoring

See above. Note that the clock is internal to the PE. The PE has no awareness of the actual timing in the monitored system, and there is no synchronization between the timing of the monitored system and that of the PE.

Forbidden element

Visual Notation

A forbidden element section optionally appears below a main chart. All elements in this section are forbidden.

Hot forbidden condition: A condition in the forbidden area surrounded with a red rectangle

Cold forbidden condition: A condition in the forbidden area surrounded with a blue dashed rectangle

Hot forbidden Message: A message in the forbidden area surrounded with a red solid rectangle

Cold forbidden message: a message in the forbidden area surrounded with a blue dashed rectangle

Vertical lines in a forbidden element are continuations of corresponding instance lines above them.

Forbidden messages with specific exact values appear exactly.

Forbidden messages that refer to all possible values of all message parameters appear as *msgname*(*) (See Fig 4.13)

When all messages between two objects are forbidden, the forbidden message appears as an arrow between the two objects, labeled with * (and no message name). (See Fig. 4.13)

When all messages are forbidden, an arrow appears across the entire LSC, labeled with *, and with no instance lines.

Each forbidden element is annotated with its scope (See below). When the scope of a forbidden element is a subchart – a dashed line connects the forbidden element and the subchart. The scope is displayed only when the mouse is over the forbidden event.

Semantics

An event or condition that has been explicitly forbidden.

Each forbidden element is associated with a level:

All: No messages allowed between any objects
Object: Messages are not allowed between the specified objects
Message: A specific message between specific objects is not allowed

Each forbidden element is associated with a scope within the LSC where it is specified:

LSC: The entire LSC
PRE: The prechart of the LSC
MAIN: The main chart of the LSC
SUB: A specific subchart within the LSC.

Forbidden Condition:

- When a hot forbidden condition is true while it is in scope, a violation occurs regardless of whether the cut was hot or cold.
- When a cold forbidden condition is true while it is in scope, the scope is exited regardless of whether the cut was hot or cold.

Forbidden Message

- When a hot forbidden message is unified with an event that occurred while it is in scope, a violation occurs regardless of whether the cut was hot or cold.
- When a cold forbidden message is unified with an event that occurred while it is in scope, the scope is exited regardless of whether the cut was hot or cold.

A hot message (or condition) is more dominant than a cold one, and a cold message (or condition) with a scope containing the scope of another cold message (or condition) is more dominant than the message (or condition) whose scope is contained. See transition for the role of dominance.

A forbidden message where the sender or receiver is a symbolic instance, refers either to the concrete object to which the same lifeline was bound, due to events in the prechart or the main chart, or, if the lifeline is not bound, to all possible objects in the class, *except* for those which appear in the chart explicitly, or were previously bound to other lifelines of the same class.

Semantics for Execution

Play-out will not select a message event if the message if either (a) the message explicitly specified as a hot forbidden message in some chart and is in scope, or (b) if the occurrence of the event will cause a hot forbidden condition that is in scope to become true.

Notes and Variants

Hot forbidden Tick messages can be used to restrict the amount of time other processes take.

Cold forbidden Tick messages can be used to limit some processes such as unbounded loops.

Violating Event

Synonym Terms

An event that violates

Visual Notation

None.

Semantics

We say that an event is a violating event, with respect to a particular cut, if there exist a strict chart in which the event appears in the main chart, but the event is not enabled in that chart at the particular cut.

Semantics for Execution

The play-out mechanism does not select violating events

Semantics for Monitoring

When a violating event occurs the run stops.

Notes and Variants

See also Hot-Forbidden Events.

Hot-Forbidden Event

Visual Notation

None.

Semantics

We say that an event is hot forbidden if is explicitly specified as a forbidden message (in the forbidden elements area) that is in scope, or it makes a hot forbidden condition that is in scope become true.

Semantics for Execution

The play-out mechanism does not select hot-forbidden events.

Semantics for Monitoring

When a hot-forbidden event occurs the run stops.

Notes and Variants

When reading or using the term hot-forbidden message/event be sure to check and clarify if the reference is to an explicitly forbidden message, or to the more general term defined here that includes also events that make hot-forbidden conditions true.

Class

Visual Notation

None. Classes per-se don't appear in charts. Only symbolic instances appear in charts.

Semantics

In LSC classes serve two purposes:

a) A class is a named collection of methods and properties. Each object is associated with a class. All the instances of this object are associated with properties and methods defined for the class.

b) The name of a class is used when specifying symbolic instances. See Symbolic Instance.

Semantics for Execution

See above

Semantics for Monitoring

Note: The classes apply only to the LSC objects and do not affect the objects of the monitored system.

Symbolic Instance

Visual Notation

An instance whose label bears a class name followed by two colons. (See Fig. 4.7, 4.14)

The label may be associated with an ellipse/oval containing a binding expression, connected with circles/bubbles to the instance header.

Solid frame and solid ellipse: universal quantifier – one or more objects;

Dashed frame and dashed ellipse: existential quantifier - a single object; (See Fig. 4.14)

Semantics

A symbolic instance is a place-holder in an LSC for a concrete instance from a class. One or more instances may be bound at run time (execution or monitoring).

A symbolic instance may be associated with a either a universal or existential quantifier and a with Boolean binding expression. The binding expression is a condition on the properties of objects in the class of the symbolic expression.

Binding occurs upon the first required event on the first of the following
- an event on the lifeline of the symbolic instance
- the beginning of the scope of a forbidden message on the lifeline of the symbolic instance
- (***) (Additional tests are needed to document all cases of binding in the Play Engine implementation)

All the variables participating in the binding expression have to be bound before binding can occur
(***) (Additional tests are needed to determine what happens when variables are not bound during a binding event)

When an event actually occurs in the system, if another LSC exists with an enabled event, or with a minimal event, that specifies a sender (receiver) as a symbolic instance that matches the class of the sender (receiver) of the actual event, then the symbolic instances are bound to the concrete objects of the actual event, respectively.

When an instance in an already activated LSC copy is bound to an object as the result of an event, the binding to the concrete objects and consequent propagation of the execution are carried out in a new separate live copy. In addition, the original copy is left open, with the cut positioned as it was in the first active live copy before the binding, but with the symbolic instance in this remaining copy being restricted in a binding expression to not bind to the same object again. In this way, prefixes of scenarios are kept open for reuse.

When binding occurs in a main chart not as a result of an actual event, if the quantifier is existential, the PE, non-deterministically, binds to the symbolic object a single object which satisfies the binding expression. If the quantifier is universal, the PE binds to the symbolic instance all objects in the class that satisfy the binding condition. If more than one object is bound, additional live copies are created.

Binding Event (of a Symbolic Instance)

Visual Notation

See Symbolic Instance

Semantics

The binding of a symbolic instance to a concrete one, is a hidden event in its own right.
See Symbolic Instance for details about when this event occurs, and its semantics.
See Transition for description of how this event participates in the execution in general.

Chart entry and exit

Visual Notation

See visual for location

Semantics

The topmost intersection of the instance line with the chart is the entry, and the lowest intersection is the exit.
The exit from a prechart is the same as the entry to the main chart of that LSC.
Entry into a main chart is always cold.
When the exit location of chart is enabled for a given instance, we say that this instance exited the chart.

Affecting a variable

Visual Notation

Variable name appears as a parameter in a message or a method call, or in LHS of an assignment.

Semantics

We say that a message affects a variable if the variable appears as a parameter of the message (property change or a method call).

We say that an assignment affects a variable if the assignment modifies (stores a new value) into the variable (variable appears on the LHS of the assignment).

Notes and Variants

We say that the variable is affected, because at the end of the event, its value may be set. Either an assignment stored a new value to it explicitly, or the variable was bound as part of unification during the execution of a symbolic property change message or a method call.

Using a variable

Visual Notation

Variable name appears as the RHS of an assignment, or as a parameter in a function call, or in an expression in a condition.

Semantics

We say that an assignment uses a variable, if the variable is part of the expression calculated in the assignment (the variable appears on the RHS of the assignment).

We say that a condition uses a variable if the variable appears in one of the expressions of the condition.

If an event uses a variable, the used variable must be bound before the execution of the event begins (it cannot become bound as part of execution of the event).

The Partial Order of Locations in an LSC

Visual Notation

See vertical order, synchronization point,

Semantics

A relationship that exists between some (not necessarily all) pairs of locations.

When the two locations are on the same instance line, then the higher one precedes the lower one.

When the two locations are in the same synchronization point – then they are simultaneous ("equal") in the partial order.

The first event that affects a variable precedes all the events that use the variable (the order of other events that affect the variable relative to the events that use the variable is not defined by this rule).

For an asynchronous message, the sending event precedes the receiving event.

In an If-Then-Else construct, the condition precedes all events in either the THEN or the ELSE subcharts and there is no order relationship between locations in the THEN subchart and the ELSE subchart.

The Partial Order is transitive – if A precedes B, and B precedes C, then A precedes C.

Notes and Variants

If one event precedes the other it does not mean that at play out they will receive different time stamps, and if two events are equal in the partial order they may still get different time stamps at play out: when in super-step mode multiple simultaneous or immediately consecutive events are executed at the same time instance. The same simultaneous or immediately consecutive events may be executed one at a time in STEP mode, and if the clock ticks in the middle of the process (e.g. if ticked manually), then they will bear different time stamps.

We say informally that one event precedes another in an LSC, if one of the locations of that event precedes one of the locations of the other event.

Event Matching

Semantics

When an event occurs or is considered for execution during the execution of an LSC, it is searched for in all LSCs in the specification. As part of the search, details of the event are compared to the details of each specified event. The search and comparison together are referred to as matching. When events are successfully matched, they may be recognized as unifiable, and then they may be actually unified. Unifiability and actual unification of events affects the triggering, satisfaction or violation of charts.

Semantics for Execution

See above

Semantics for Monitoring

Events emanating from the monitored system, and events generated by the LSCs in the specification, are matched to the LSC specification.

Unification

Semantics

When event matching determines that two events are, or can be, the same event, the two events are considered unifiable. Depending on the situation, unifiable events may be subsequently unified.

.

Positive event unification is the successful matching of an event from a monitored system, or of an LSC event, with an event which is either minimal in a prechart, or is enabled in a main chart or in an existential chart, for the following purposes:

- If the second event is a minimal event in a prechart or in an existential chart, then positive unification changes the mode of the second event's LSC to preactive or to check, respectively.

- If the second event is an enabled event in an LSC then the second event will be advanced following the interception of the first event if it was a monitored system event, or simultaneously with the occurrence (=execution) of the first event, if it was an LSC event.

In general, for two events to be unifiable the following must hold:

- If the two senders (respectively, receivers) are concrete, they must be the same concrete objects

- If one sender (respectively, receiver) is a concrete instance and the other sender (respectively, receiver) is a symbolic instance, then both instances must be in the same class and the concrete object cannot be forbidden in the binding expression of the symbolic instance. The symbolic instance is then bound to the same object as the concrete instance.

- The messages must be unifiable, which means that:

    - If one message is a property change, so must be the other, and the names of the properties must be the same.

    - If one message is a method call, so must be the other, and the names of the methods must be the same.

    - Corresponding property values, variables and function specifications used in parameters for property changes or method calls must be unifiable as follows:

        - Corresponding exact property values and exact method call parameters must be equal.

        - If X and Y are corresponding variables then either:

            - X and Y are both bound and their values are the same. (If X and Y are both bound and their values are different - they cannot be unified)

            - X is bound and Y is free. Y is then bound to the same value as X.

            - X and Y are both free. X and Y are then placed in the same connection set (see comment under connection set – in practice this situation is not allowed)

        - Corresponding function specifications are unified if all parameters of the functions are bound and the values of the functions are equal.

        - A variable and a corresponding function specification are unifiable, if (a) all parameters to the function are bound, and (b) either the variable is bound to the same value as the function specification, or the variable is free (it is then bound to the value of the function).

Negative event unification is the successful matching of an event from a monitored system, or of a pending LSC event, with an event which appears in an active or preactive strict chart where it is not enabled, or is forbidden in some chart.

In general, for two events to be negatively unified the same conditions as for positive unification must hold, with the following exceptions:

- Unified variables cannot both be free

- The event of method call Tick, is never negatively unified.

For cases where the second event is a forbidden event, see forbidden elements.

If the second event is a non-enabled event in a main chart of a strict chart, then
if the first event is a pending LSC event, it is deferred.
Else (the first event emanated from a monitored system, or otherwise occurred already)

If the second chart is in a cold cut – the second chart is abandoned,

Else (the second chart is in a hot cut) a [hot] violation occurs (see under violation. See more details under Transition and step).

In general – successful negative unification may cause the suspension of processing of an LSC. In execution, if no other LSC can proceed, the entire system is suspended (see step procedure for details)

Notes and Variants

Unification can be at different levels:

Level0: The messages are constant/exact
Level1: The messages may include constants and variables
Level2: The messages may include constants, variables and symbolic instances

Reachable Event

Semantics

The concept of an event which is reachable from a cut C is defined recursively:

- The event must be a visible event
- It is either an enabled event, or
- It is reachable from the cut yielded by transitioning from the current cut thru an enabled hidden event, or
- It is reachable from the end of a dynamic loop whose start is enabled, or it can be reached from the start of a dynamic loop, whose end is enabled.

Transitions advance cuts to a next reachable event.

Transition

Semantics

A separate section in this document describes the general transition rules. The reference entry for each LSC construct contains specific execution rules required for the transition.

Minimal event

Semantics

An event in the prechart of a universal chart, which is visible, and is minimal in the partial order induced by the LSC (it is not preceded by other visible events). It may be preceded by hidden events ("first" in graphs).

Anti-scenario

Semantics

An LSC that causes a (hot) violation whenever its prechart is satisfied. The anti-scenario is not marked as being an anti-scenario, but is described as such as a result of how it is programmed. For instance – the only event in the main chart is a hot FALSE condition.

Semantics for Execution

During execution under smart play-out anti-scenarios are avoided by the play engine. If they cannot be avoided (e.g., forced by external events) - a violation is indicated. Note that the violation is in the main chart - and causes system abort.

Semantics for Monitoring

Anti-scenarios cannot be avoided or delayed. When their prechart is satisfied – a violation occurs.

Activation Modes and Life Cycle of a Live Copy of an LSC

Semantics

When minimal events in a prechart of a universal chart or in an existential chart are matched, a live copy of the LSC is created. The mode of a live copy of a universal LSC may be either preactive - when in prechart, or active - when in main chart.

A preactive live copy of LSC may either become active - when all its instances reach the end of the prechart, or it may end its life - when it is abandoned due to a "cold violation" (=prechart violation).

An active live copy of an LSC may end its life by being either
- Completed - when all instances reach the end of the chart.
- Hot-violated (See definition of violation)
- Cold violated / Abandoned (See definition of violation)

The mode of a live copy of an existential chart may be either check mode, or completed. It may end its life by being either:

- completed
- cold violated

Notes and Variants

The term *terminate* is also sometime used for the completion of live copies of LSCs.

Connection set

Semantics

When two free variables are positively unified, they both remain free, and are connected as part of a connection set. When one of the variables in a connection set is bound, all other variables in this connection set are bound to the same value.

Notes and Variants

The concept of a connection set enables, in theory, the happening of symbolic events without their variables being bound. The current implementation of the PE disallows this situation, and prompts the user to specify values for free variables that cannot be resolved by unification with other events.

Scope of an LSC construct

Visual Notation

Defined by instance lines with which the construct intersects - and whether it hides them or not, and if not, whether the intersection is marked by a connector or not.

Semantics

The concept of scope is defined separately to some LSC elements such as assignment, condition, if-then-else, loop, subchart.

SynchronizationPoint

Visual Notation

See visuals for subcharts, conditions, loops, assignments, if-then-else;

Semantics

A collection of locations from different instances which are subjected to the requirement that none of these instances can proceed past its location in the synchronization point until all other instances reached their locations in the synchronization point.

Notes and Variants

The synchronization point is marked by intersection of an instance line with a construct. Details of the marking varies by construct: Main charts use plain intersection, assignments and conditions use connectors, subcharts (including if-then-else constructs and loops) use “showing through” versus “hiding”.

Subchart

Visual Notation

A black, thick, solid rectangle. Instance lines may be visible in it, or may be hidden "behind" it. (See Fig 4.12)

Semantics

The subchart provides synchronization points for all its participating instances (lines visible thru it). They all enter it and exit it together.

Forbidden conditions may be associated with a subchart as their scope. .

Instances that are hidden by the subchart do not participate in it.

Current Chart

Visual Notation

See visuals for all chart types

Semantics

In a given running LSC, for a given instance, the subchart that directly contains the current location. If there are nested subcharts - it is the innermost that contains the current location. If there are no subcharts -the current chart is the prechart or main chart containing the location.

Strict LSC

Visual Notation

See visual for LSC.

Semantics

In a strict LSC, if an event that appears in the LSC actually happens when it is not enabled, hot or cold a violation is indicated (depending on the temperature of the cut).
(See Tolerant LSC).

Semantics for Execution

During execution, events that may cause a hot violation of a strict LSC as described above are not executed.

Tolerant LSC

Visual Notation

Word “tolerant” appears next to LSC name

Semantics

In a tolerant LSC, if an event that appears in the LSC actually happens when it is not enabled, no violation is indicated (See Strict LSC). Events can violate a tolerant LSC only if they are specified as forbidden using the forbidden elements mechanism.

Environment

Visual Notation

A cloud with the word "ENV" . (See Fig 4.7)
The instance line of the object ENV (environment) is comb-like.

Semantics

A predefined object which represents the environment (real world) outside of the system (executed or monitored). It can send and receive messages/events.

Object System

Visual Notation:

None

Semantics

A set of objects, classes, types, implemented functions, and a clock. This term is used as part of the formal definition of LSC.

Play Engine Terms

Play Engine

Visual Notation

Play engine software screens with menu and work area

Semantics

Semantics for Execution

The Play Engine (PE) is used for specifying LSCs through play-in, and for executing them through play-out.

Semantics for Monitoring

The PE is used for specifying LSCs through play-in, and for monitoring systems through play-out.

Notes and Variants

During play-out – the PE user specifies which LSCs will be executed and which will be monitored.

GUI Application

Visual Notation

A collection of buttons and visual artifacts, serving to represent the System. Example – calculator app.

Semantics

Semantics for Execution

The GUI application is used by the play-engine during play-in as a tool for directly specifying elements such as user actions, object state changes, or selecting objects for synchronization. In play-out, it provides more intuitive visualization of certain property changes.

Semantics for Monitoring

During monitoring the GUI app doesn't have a role.

Internal Object

Visual Notation

Same as object

Semantics

Internal objects are objects that can be fully controlled by the PE.

External Object

Visual Notation

An instance annotated with a small cloud

Semantics

External objects have the same semantics as an internal object, with the exception that during play out, whether in monitoring or execution, the PE user fully controls any actions taken by external objects – such as changing properties, or sending messages. The PE does not initiate the sending of messages from these objects.

Implemented Function

Synonym Terms

Function

Visual Notation

The function appears as part of a Right Hand Side of an assignment or condition.

Semantics

A process/algorithm/computation that uses property values, variables, and constants to create another value.

Function calls can be used in Conditions and in Assignments.

During play out the PE calculates the value and uses it as a constant.

Semantics for Execution

A function can be used to augment the processing power of the LSC specification towards generating computational results.

Semantics for Monitoring

The values of the function can be used for controlling the flow of the PE, or for validating results from the monitored system. The function does not participate in producing computational results of the system.

Notes and Variants

Functions cannot have side-effects beyond the local scope and time of each evaluation.

When the parameters to a function are all constants, the function is evaluated once during play-in, and is treated as a constant.

Play In

Semantics

Process of drawing (specifying) an LSC either by explicit specification or by manipulating objects in the GUI application.

Use Case

Visual Notation

Semantics

A collection of LSCs.
A mid-level hierarchical artifact that helps in organization of LSCs in the PE. It does not carry execution semantic.

User

Visual Notation

A stick drawing of a person

Semantics

One of the objects in the system. Events are associated with this object, during play-in and play-out, when the PE user manipulates the controls of the GUI application.

Semantics for Execution

The user represents external actions that can be entered by the PE-user on the mockup GUI application

Semantics for Monitoring

The user is just another object of the monitored system, which can generate events. Typically refers to a human being.

Property attribute “In only"

Semantics

Indicates that this property can only be changed by a user action, and not by the system. This specification helps prevent creation of invalid LSCs.

Property attribute: "Can be changed externally"

Semantics

Property value can be changed by the environment. This specification allows control of creation of valid LSCs.

Property Attribute: "Affects"

Visual Notation

See "Message"

Semantics

Affects can have the value of either User, Self, or Env. When set, when the value of the property is changed during play-in, an arrow is drawn in the PE from the object at hand to the appropriate receiving object, which is either itself, the user or the environment.

Property Prefix

Semantics

The verb/method that is associated with the action of changing the property. It is used for clearer action notation in charts. E.g. it allows a message for turning on a light object to be labeled "Turn(on)" rather than require it to refer to the property called State, and be specified as "State(on)".

Property attribute: "Is default"

Visual Notation

On the chart, the property values are associated with the object name, rather than with the property name

Semantics

The default property whose value can be said to be the value of the object. E.g. The object is "on" (for simplification of English sentences).

Property attribute "Synchronous"

Visual Notation

Visually represented by incoming message being synchronous

Semantics

All messages affecting this property are synchronous.

String expression

Visual Notation

A user-entered string as one of the conjuncts in a condition. Normally text describing a question or a condition which may or may not be true.

Semantics

In a condition, the user can specify an arbitrary string expression as a conjunct. During play-out, the PE will display the string and prompt the user to specify whether this condition is true or not.

3. LSC Processing – Naïve Play Out

Introduction

This section provides an algorithmic description of the following aspects of play-out semantics:

- The transition procedure – processing of a single event, in execution or monitoring.
- The monitor-event procedure – applying the transition above to monitoring
- The step and super-step procedure – applying the transition above to executing an LSC specification

The following principles are used in the pseudo-code below

- Minimal use of math and programming notation or of unique definitions that are outside of the procedure – to enable continuous reading by a broad audience.
- No usage of special fonts or subscripts.
- Reliance on the rest of the reference document for details of processing of individual constructs, and/or for description of synchronization.
- The steps are processed sequentially – if-statements are processed as in a regular program (when the if-condition is false, and there is no else-clause, processing continues in the next sequential statement).
- The term *advance* as used here, means changing the current cut into a new cut, by changing the current location in one of the instance lines, to another location in the instance line. It does not imply doing anything else associated with the semantics of the event. With a few exceptions, changes to property values are done in the step procedure, outside of the transition procedure.

Transition Procedure

The following pseudo code program describes the transition procedure defined above. The main input to the process is an event e1 selected by the monitor-event or the step procedures. The transition procedure accesses the LSC specification, a current collection of live copies of LSCs, and, indirectly, the set of objects and their properties and variables, with their values and binding status.

1. If e1 is the sending or the receiving of a message, then:
   1.1. If e1 is a clock tick, then increment the property Time of the object Clock.
   1.2. [Empty step – to retain the matching to the book's numbering)
   1.3. If e1 is unifiable with at least one forbidden message in one live copy of an LSC, and there is no enabled event e2 such that e1 and e2 are unifiable then:
      1.3.1. Pick one of the forbidden messages of the highest priority ("dominant"), and process it per its temperature, level and scope. (Where applicable, cause a violation or exit a current chart/subchart)
      1.3.2. (***) (Requires checking - should be for ALL forbidden msgs)
   1.4. If e1 may cause a violation (is negatively unifiable with an event e2 which is not enabled, in some live copy of an LSC, called CL1(but is not forbidden, as forbidden messages were handled above), then:
      1.4.1.If the current cut is hot, then indicate a violation.
      1.4.2.If the current cut is cold:
         1.4.2.1. If the sending object in e2 is a symbolic instance, SI1, then replicate CL1 in a new live copy, say, CL2, and indicate in it, that SI1 cannot be bound with the concrete corresponding sending object in e1. Then, do the same for the receiving object in e2.
         1.4.2.2. Abandon/terminate the live copy CL1
   1.5. If e1 is unifiable with a minimal event e2 in any prechart of any universal LSC, then, for each of these LSCs:
      1.5.1.Create a live copy for that LSC; set its mode to preactive; set its cut past the start location in that lifeline(s) where e2 appears.
      1.5.2.Unify e1 and e2 (including symbolic instance binding and variable binding – see ref.)
      1.5.3.Apply this transition procedure to each hidden event e3 that precedes e2 in the new live copy.
   1.6. If e1 is unifiable with a minimal event e2 in any existential LSC, then, for each of these LSCs:
      1.6.1.Create a live copy for that LSC; set its mode to check; set its cut past the start location in the lifeline(s) where e2 appears.
      1.6.2.Unify e1 and e2 (including symbolic instance binding and variable binding – see ref.)
      1.6.3.Apply this transition procedure to each hidden event e3 that precede e2 in the new live copy.
   1.7. If e1 is positively unifiable with a reachable event e2 in any live copy of an LSC, CL1, then for each such CL1:
      1.7.1.If the sending object in e2 is a symbolic instance, SI1, then replicate the live copy CL1 (including its current cut) in a new live copy, say, CL2, and indicate in it that SI1 cannot be bound with the concrete corresponding sending object in e1. Then, do the same for the receiving object in e2.
      1.7.2.Advance past the event e2 in CL1 (see ref.)
      1.7.3.If e2 is the sending of a synchronous message, advance past the corresponding receiving event (the symbolic instance binding associated with the receiving object is handled below).
      1.7.4.Unify e1 and e2 (including symbolic instance binding and variable binding – see ref.)
      1.7.5.Apply this transition to all hidden events that precede e2 in CL1 (this is not a recursive call – the reachable events are not hidden).

2. If e1 is the completion of a live copy of an LSC, CL1, then abandon/terminate this live copy.
3. If e1 is the completion of a prechart of a live copy of an LSC, CL1, then set the cut to be at the beginning of the main chart, and change the mode of CL1 to be active.
4. If e1 is an assignment construct, then:
   4.1. Perform the assignment (see ref.)
   4.2. Advance past the event
5. If e1 is an evaluation of a condition, then:
   5.1. Evaluate the condition (see ref.)
   5.2. If the value is true, advance past the condition (see ref.)
   5.3. If the value is false, then:
      5.3.1.If the condition is hot, then indicate a violation**.**
      5.3.2.If the condition is cold, exit the current chart (that is, If-then-else construct, subchart, or LSC (See ref.))
6. If e1 is the evaluation of a condition in an If-then-else construct, then:
   6.1. If the condition is true, advance into the "then" part of the construct.
   6.2. If the condition is false, then:
      6.2.1.If there is an "else" part to the construct, then advance into it,
      6.2.2.If there is no "else" part, then advance past the end of the if-then-else construct.
7. If e1 is the end of the "then" part of an if-then-else construct, advance past the end of the if-then-else construct.
8. If e1 is the end of the "else" part of an if-then-else construct, advance past the end of the if-then-else construct.
9. If e1 is the end of any subchart, advance past the end of the subchart
10. If e1 is an indication to skip a dynamic loop, then advance past the end of the loop
11. If e1 is the beginning of a loop body, then:
    11.1. If the loop is dynamic – prompt the user for the loop control number), and set it.
    11.2. Advance into the loop.
    11.3. Reset the value of any variable used for the first time in the loop, to be empty (undefined, unbound) (see ref.).
12. if e1 is the end of a loop, then:
    12.1. Decrement the loop control number (assuming that it was set as part of the definition of the loop, or was supplied at the initialization of a dynamic loop).
    12.2. If the loop control number is greater than zero, then
       12.2.1. Advance to the beginning of the loop
       12.2.2. Reset the value of any variable used for the first time in the loop, to be empty (undefined, unbound) (see ref.)
    12.2.3. If the loop control number is zero, then advance past the end of the loop
13. Is e1 is a binding event for a symbolic instance SI1 in a live copy of an LSC CL1, then
    13.1. Examine all the concrete objects in the object system that are of the same class as SI1, satisfy the binding conditions of SI1 and are not yet bound.
    13.2. If no such objects exist, abandon/terminate CL1.
    13.3. If such objects exist then:
       13.3.1. If CL1 is existential, then bind SI1 to an arbitrary concrete object from the above
       13.3.2. If CL1 is universal, then:
          13.3.2.1. For each concrete object from the above, replicate CL1 to a new live copy CL2, and bind SI1 to that concrete object.
          13.3.2.2. Abandon/terminate CL1

14. If e1 is the evaluation of a forbidden condition (this evaluation in the Transition procedure is invoked only following an evaluation by Monitor-Event procedure, which resulted in true; therefore the condition must be true), then:
    14.1. If the condition is hot, then indicate violation.
    14.2. If the condition is cold, abandon the current subchart or chart (see ref.)

Monitor-Event Procedure

During monitoring (testing) the PE receives a stream of events emanating from the monitored system. This may happen in real time, or by processing a log/trace file of a sequence of events that occurred in the monitored system. The events are processed sequentially, and are passed, one-at-a-time, to the Monitor-event procedure below.

During execution (simulation), the Step procedure calls the Monitor-Event procedure as part of the processing of the event.

Note: To match the flow of Monitor-Event in “Come Let’s Play” book, the pseudo code below uses “go to” instead of while (or repeat-until) loops.

1. Process the event using the transition procedure above.
2. Create a collection of all forbidden conditions that are true, and whose scope is active.
3. If the above collection is not empty, then:
    a. Pick the condition which is of highest priority (most dominant) of the remaining ones.
    b. Apply the transition procedure above to the event of evaluating this forbidden condition.
    c. Go back to step 2 above
4. Create a collection of all enabled events, excluding any events that are a branching event in an If-then-else construct, or the end of a dynamic loop.
5. If in the above collection there are hidden events, then
    a. Pick one such enabled hidden event, and apply to it the transition procedure above
    b. Go back to step 2 above.

Step Procedure

During execution, the play engine picks the next event to be processed as follows:

1. Apply the transition procedure to the selected event.
2. If the event was a change to an object property, then
    a. If the destination of the message is the User or the Environment, then the affected object is the message source. For all other messages, the affected object is the message destination.
    b. Change the property of the affected object as follows:
        i. If the new value is a function specification – compute it, and assign the value of the function.
        ii. If the new value is provided explicitly, assign the provided value.
3. Call “Monitor-event” procedure

Super-Step

In a super-step, the PE executes a series of steps (as described above) associated with internal events. Internal events are those that don’t originate from the user, the environment, the clock, or external objects. The execution by the PE continues until no more such events can be carried out.

Note: The following super step procedure applies a maximal policy.

1. Create a collection of all forbidden conditions that are true, and whose scope is active.
2. If the above collection is not empty, then:
    a. Pick the condition which is of highest priority (most dominant) of the remaining ones.
    b. Apply the transition procedure above to the event of evaluating this forbidden condition.
    c. Go back to step 1 above
3. Create a collection of all internal enabled events, excluding any events that are a branching event in an If-then-else construct in a prechart.

4. If in the above collection there are hidden events, then
    a. Pick one such enabled hidden event, and apply to it the transition procedure above
    b. Go back to step 1 above.
5. If there is an enabled visible event in the above collection, that also does not violate , and is neither hot-forbidden, nor cold forbidden in any copy which is either in active mode or preactive mode, then:
    a. Execute the procedure Step above with this event.
    b. Go back to step 1 above
6. If there is an enabled visible event in the above collection, that also does not violate, and is neither hot-forbidden nor cold forbidden in any copy which is in active mode, then:
    a. Execute the procedure Step above with this event.
    b. Go back to step 1 above
7. If there is an enabled visible event in the above collection, that also does not violate, and is not hot-forbidden in any copy which is in active mode with a hot cut, then:
    a. Execute the procedure Step above with this event.
    b. Go back to step 1 above

4. Visual Notation Examples

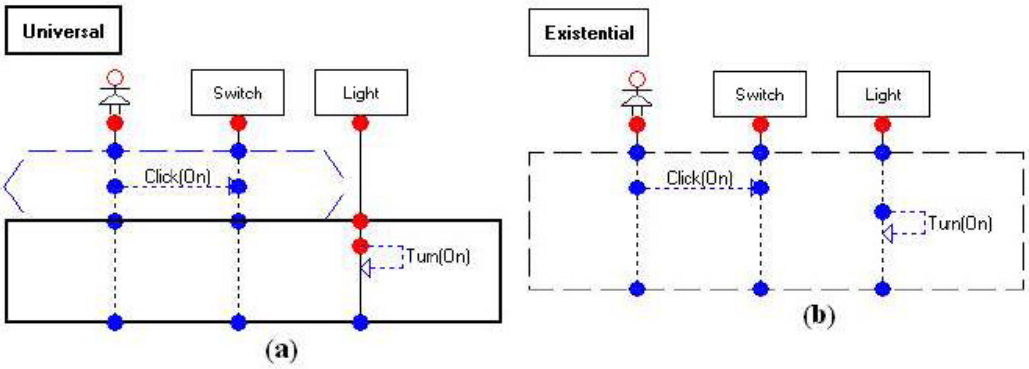


Figure 4.1.

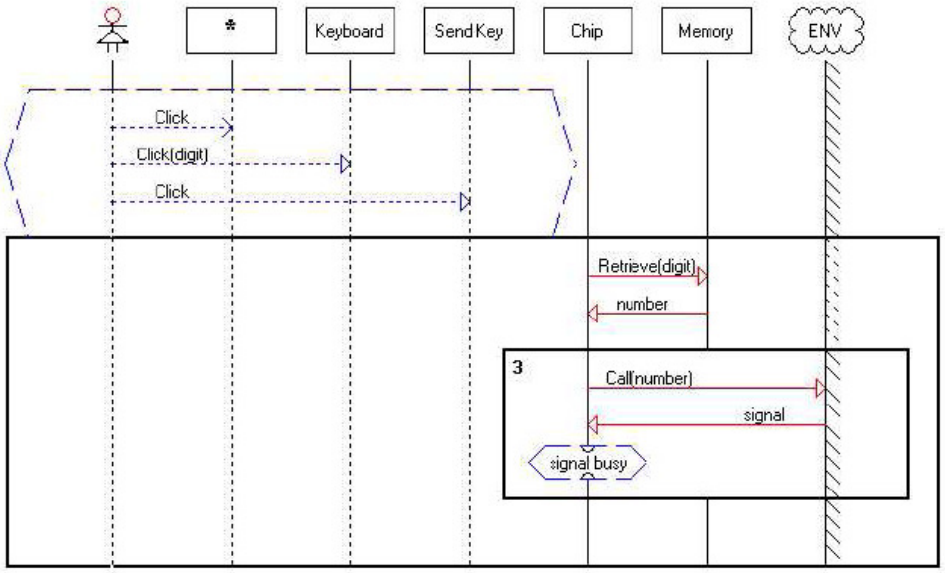


Figure 4.2

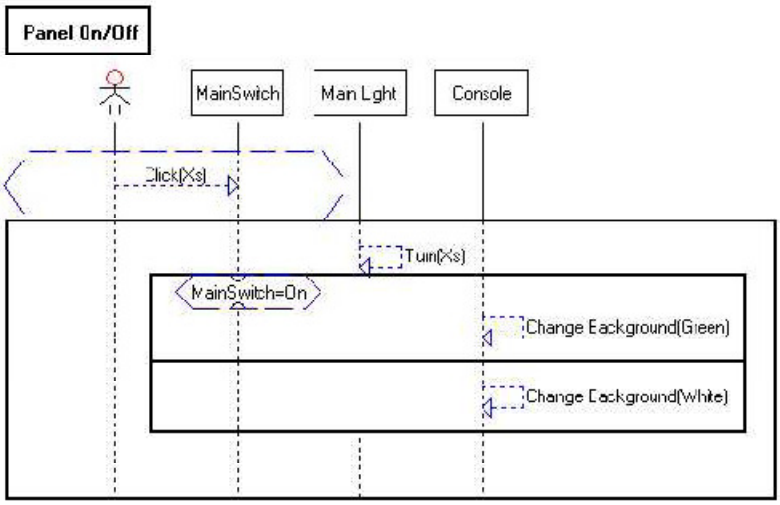


Figure 4.3

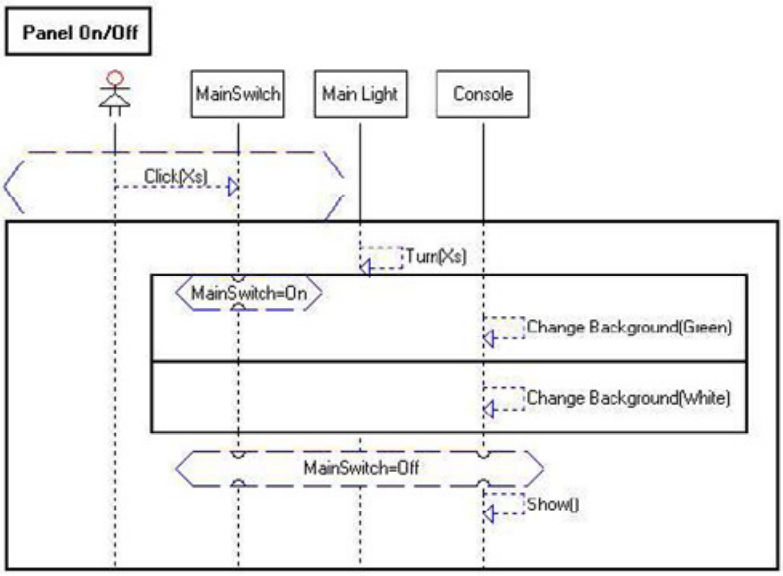


Figure 4.4

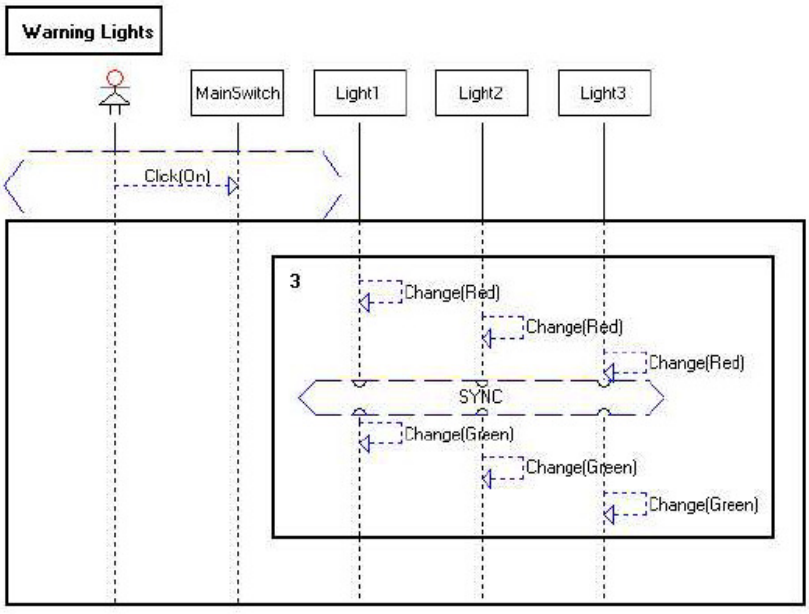


Figure 4.5

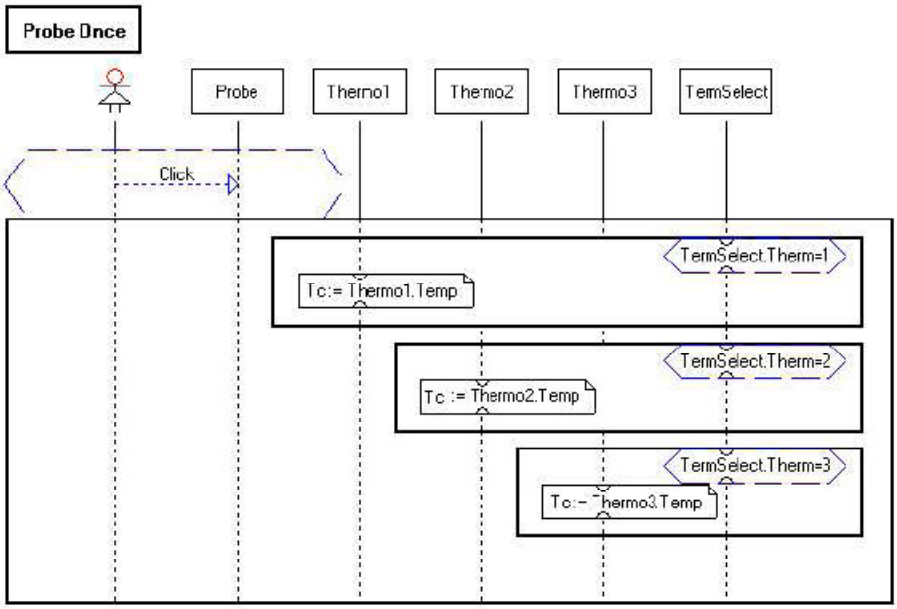


Figure 4.6

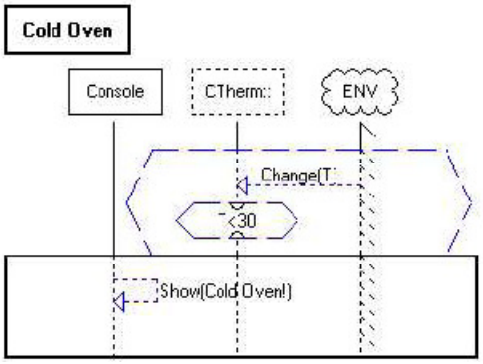


Figure 4.7

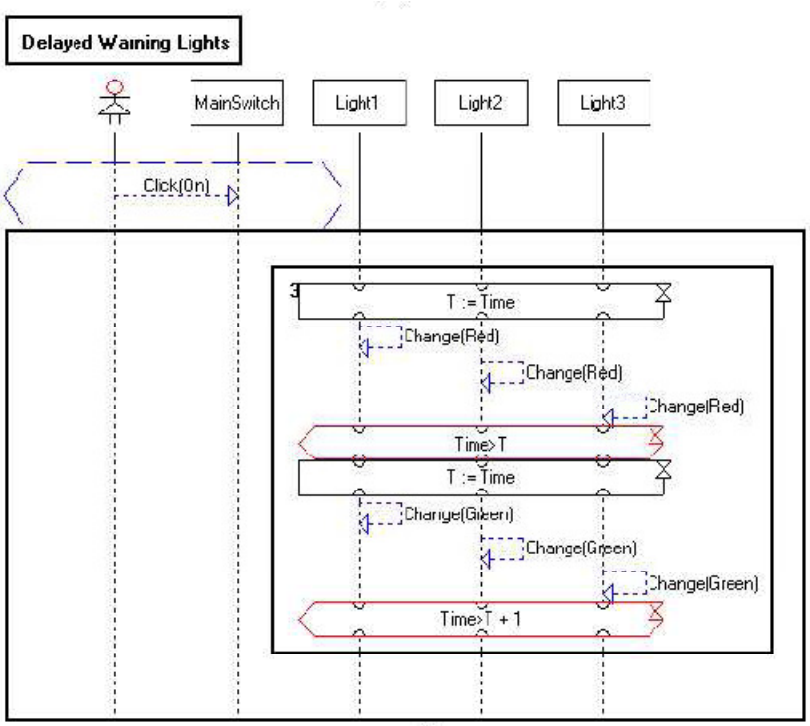


Figure 4.8

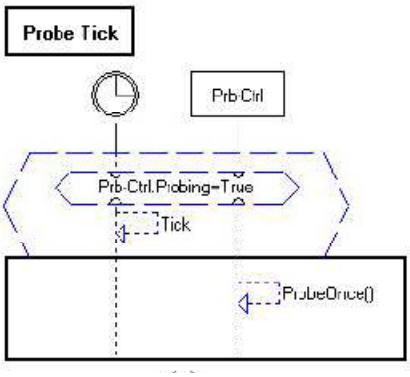


Figure 4.9

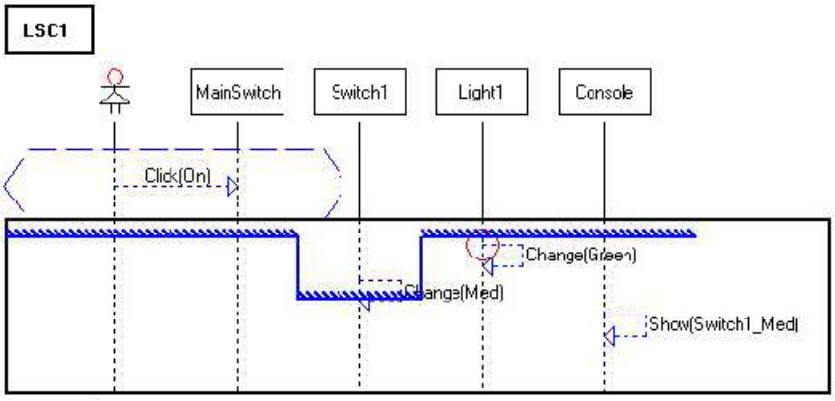


Figure 4.10

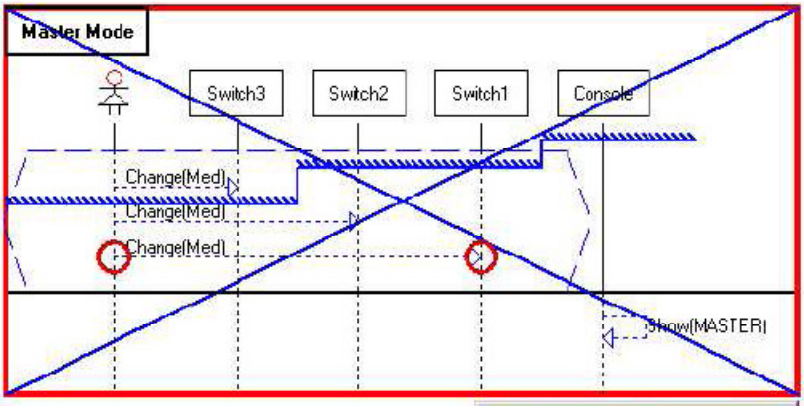


Figure 4.11

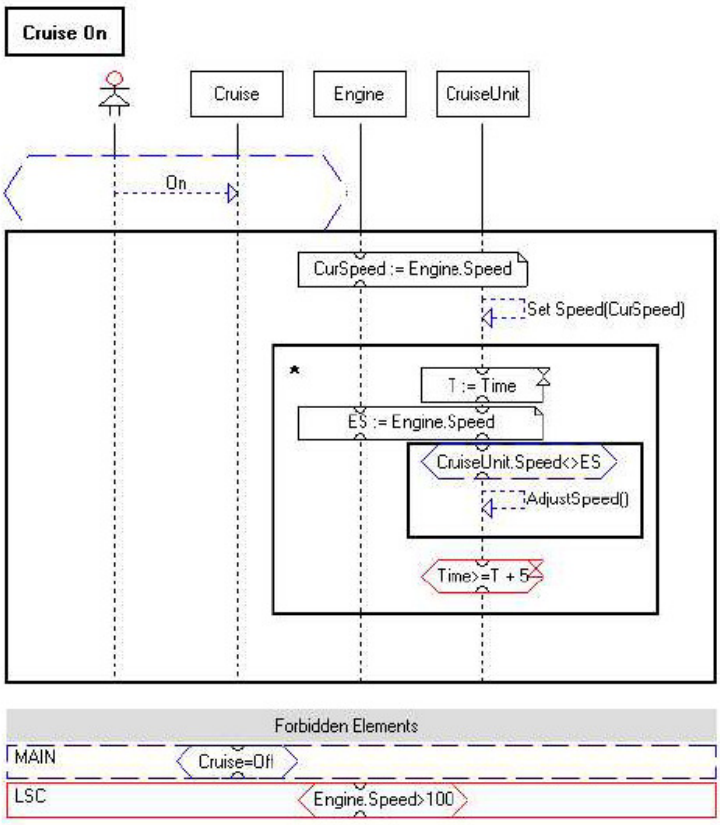


Figure 4.12

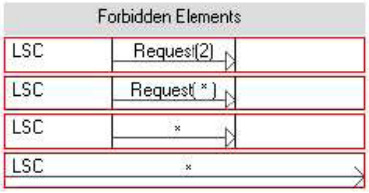


Figure 4.13

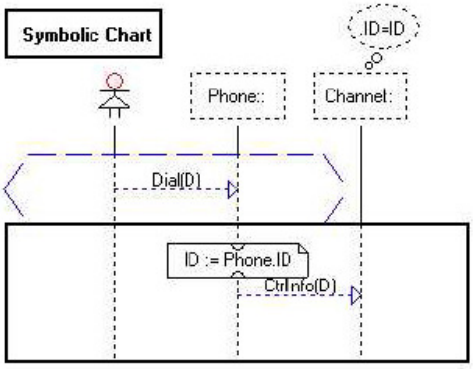

Symbolic Chart
ID=ID
Phone::
Channel:
Dial(D)
ID := Phone.ID
CtrlInfo(D)

Figure 4.14

References

1. Damm, W., Harel, D. Breathing Life into Message Sequence Charts. Formal Methods in System Design 19(1) (2001) (Prelim. ver.: IFIP FMOODS'99, Kluwer,1999).

2. Harel, D., Marelly, R.: Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer-Verlag (2003)

3. Harel, D., Marelly, R.: Play-Engine User's Guide. Weizmann Inst. of Science (Oct. 2002)

4. Harel, D.: List of Publications. (Accessed 2009) Available at: http://www.wisdom.weizmann.ac.il/~harel/papers.html

5. Harel, D.: Can Programming be Liberated, Period? IEEE Computer 41(1), 28-37 (2008)

6. Harel, D., Pnueli, A.: On the Development of Reactive Systems. In : Logics and Models of Concurrent Systems. NATO ASI Series, Vol. F-13. Springer-Verlag, New York (1985)

7. ITU-TS: ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). (1996)

*