Abstract: Cost over-run and production delay have been serious problems faced by software developers for the past several decades. One way of mitigating this is the use of pictorial symbols instead of text for expressing the design and implementation details involved in software development. As we say, a picture is worth a thousand words and one such representational system which uses this aphorism in actual practice is the Specification and Description Language. Pictorial representations, being self-explanatory, naturally reduce the costs incurred for documentation. Moreover, most of these symbols can be efficiently used for the auto-generation of some of the basic functionalities thereby reducing the production time. This is especially useful because it is not possible for a software developer to keep track of the continuously evolving and newer software languages. So the pictorial representation comes in handy because of its capacity for the said code auto-generation! We feel that the most efficient way of doing this is using symbols confirming to Specification and Description Language - Real Time (SDL-RT) standards.

1. Introduction: SDL is an object-oriented high-level programming language that is formal and graphical in nature. It is designed for the description of complex, event-driven systems capable of communicating in real time.

SDL development was originally initiated by International Telecommunication Union (ITU) in 1972. The ITU has been launching newer versions of this language every fourth year or so since the first launch in 1976, the latest being in 2000. SDL diagrams are easy to understand by laymen and software developers alike, thanks to its highly graphical character. This results in greatly enhanced interaction between the system designer and the client. It is for this reason that SDL can be efficiently used to auto-generate executable applications, leading to reduced time and cost to develop the desired software product. Due to the readability of SDL diagrams, its specification becomes documentation in itself, thereby generating efficient documents and reducing the cost and time required for documentation during software development.

Specification Description Language - Real Time (SDL-RT) is based on SDL standards developed by ITU, extended with real time concepts. It is designed
for the real time developer looking for graphical representations of traditional concepts used in his domain, without leaving behind a legacy code and years of experience. SDL-RT is based on tasks running concurrently that exchange information through messages and synchronizes with each other using semaphores.

SDL has its own data types and syntax whereas SDL-RT basically uses ANSI C language. Some symbols have a specific syntax with SDL-RT since there is no C equivalent instructions such as output, input, save, or semaphore manipulations.

2. SDL-RT Standards: The following sections discuss the symbols used in SDL-RT that can be used in the development of the desired software product.

System: The overall design of the software product is called the “system”. The universe outside the system is called the environment.

Agent: An agent is a component in the system structure. There are two kinds of agents namely, “blocks” and “processes”. Blocks are the primary divisions of the system. A block can be further divided into sub-blocks and so on. Dividing the system into blocks thus enables the pictorial representation of all the essential components of the given system. A block is represented by a solid rectangle with its name in it.

![Block Symbol](image1)

Figure 1: Block

When the system is divided down to the simplest block, the way that block fulfils its functionality is described by “processes”.

A process is represented by a rectangle with cut corners with its name in it:

![Process Symbol](image2)

Figure 2: Process Symbol

A process is basically the code that will be executed. It is possible to have several instances of the same process running independently. The full syntax in the process symbol is:

```plaintext
<Process name> [(<number of instances at startup>, <maximum number of instances>)]
```

If omitted, the default values are as follows: 1 for the number of instances at startup and infinite for the maximum number of instances.

SDL-RT is based on an event driven architecture that accomplishes the required task through an exchange of messages. A message has a name and a parameter that is basically a pointer to some data. Messages go through
channels that connect agents and end up in queues implicit to every process.

Channels have names and are represented by one-way or two-way arrows. A channel name is written next to the arrow and the list of messages going in a specific way are listed next to it between brackets, separated by commas. A channel end point can be connected to: the environment, another channel or a process.

![Figure 3: Channel Symbol](image)

The beginning of the process is represented by the following symbol:

![Figure 4: Start Symbol](image)

A process can terminate itself which can be represented by:

![Figure 5: Stop Symbol](image)

Messages move between different agents: The Input message for any particular agent is represented by:

![Figure 6: Message Input Symbol](image)

Whereas the output message coming from any particular agent is represented by:

![Figure 7: Message Output Symbol](image)

A process may be multi-step and then it may not be able to handle newer requests until the on-going job is complete. It is therefore imperative that this new information be saved until the process reaches a stable state. This information that needs to be saved is represented as:

![Figure 8: Save Symbol](image)

There are several times wherein the flow of control in the process depends on certain conditions. Such conditions are represented using the following symbol.

![Figure 9: Condition Symbol](image)

The program control flow is determined by certain variable conditions. The possible options or paths that might be used are thus represented:

![Figure 9: Decision Symbol](image)
The values of the branches have keyword expressions such as:

- `>`, `<`, `>=`, `<=`, `!=`, `==`
- `true`, `false`, `else`

Semaphore take: The “Semaphore take” symbol is used when the process attempts to take a semaphore.

Figure 10: Semaphore Take Symbol

Similarly, the “Semaphore give” symbol is used when the process attempts to leave the semaphore.

Figure 11: Semaphore give Symbol

Timer Start: A Timer start symbol represents the timer used to set the time for a certain event to take place in terms of clock ticks.

Figure 12: Timer start symbol

New process creation: A new process to be created is represented using the following symbol.

Figure 13: New Process creation Symbol

To create several instances of the same process which may run concurrently, we use the following symbols:

Figure 14: Instances of a single process

Connectors: Connectors are used to split a transition into several pieces so that the diagram stays legible and printable. A connector contains a name that has to be unique in the process.

Figure 15: Connector Symbol

Comment: The comment symbol is used to write any type of informal text and connect it to the desired symbol.

Figure 16: Comment Symbol
Extension symbol: The extension symbol is used to complete an expression in a symbol. The expression in the extension symbol is considered part of the expression in the connected symbol. Therefore the syntax is the one of the connected symbol.

Figure 17: Extension Symbol

Example:

Procedure start symbol: It indicates the procedure entry point.

Figure 18: Procedure start symbol

Procedure return: indicates the end of the procedure.

Figure 19: Procedure return symbol

Text Symbol: is used to declare C-type variables.

Figure 20: Text Symbol

Object Creation Symbol: represents a pictorial way of creating objects. This is equivalent to creating an instance of class \(<\text{class name}>\) named \(<\text{object name}>\).

Figure 21: Object Creation Symbol

Time interval: To specify a time interval between two events the following symbol is used.
Co-region: is used whenever the sequence of events does not matter. Events in a Co-region can happen in any order.

Class: A Class describes the set of objects similar in structure, behavior and relationships. It is represented by:

Package: serves as a container for agents or classes. They are similar to packages present in Java and are represented by the following symbol.

3. Conclusion: We feel one of the possible ways of mitigating delays and cost over-run during software development is to use pictorial symbols for expressing the design and implementation details involved in software development. Pictorial representations, being self-explanatory, naturally reduce the costs incurred for documentation. Moreover, most of these symbols can be efficiently used for the auto-generation of some of the basic functionalities thereby reducing the production time. We feel that the most efficient way of doing this is using symbols confirming to Specification and Description Language- Real Time (SDL-RT) standards.
4. References:
5. Specification and Description Language - Real Time (www.sdl-rt.org)
6. Kevin Strike, Khaled El Emam and Nazim Madhavji “Software Cost Estimation with Incomplete Data”
8. Serge Oligny, Pierre Bourque, Alain Abran and Bertrand Fournier “Developing Project Duration Models in Software Engineering”
10. Frank Padberg and Fakult“at f”ur Informatik.”Applying Process Simulation to Software Project Scheduling”
11. Sarah Jane Delany1, Pádraig Cunningham2 and Wolfgang Wilke3“The Limits of CBR in Software Project Estimation”
12. Kadoda, Michelle Gada Cartwright, Liguang Chen, and Martin Shepperd“Experiences Using Case-Based Reasoning to Predict Software Project Effort”
13. Warren Rost Bennett, Jr. “Predicting Software System Development Effort Very Early in the Life Cycle using Idef0 and Indef1X models”