A comparison of Unified Modelling Language (UML) and Specification and Description Language (SDL)

Marshini Chetty

02-01-00
March 17, 2002

Data Network Architectures
Department of Computer Science
University of Cape Town
Private Bag, RONDEBOSCH
7701 South Africa
e-mail: dna@cs.uct.ac.za

1 Introduction

This report serves as a basic introduction to the formal specification languages, UML and SDL.

1.1 What is UML ?

According to Booch et al [1], UML (Unified Modelling Language) is a standard language for visualizing, specifying, constructing and documenting the artifacts of a software system. It is an object oriented language with a graphical representation and is standardised by the Object Management Group (OMG). UML can be used to model many different types of software and other systems.

1.2 What is SDL ?

SDL (Specification and Description Language) is a standard language used to specify and describe the functional behaviour of telecommunication and other systems. SDL is standardized by ITU (International Telecommunications Union), as standard Z.100. It has a both a graphical format (SDL/GR) and a textual format (SDL/PR) [2].

More specifically, an SDL specification models systems with concurrently running processes. These processes are finite state machines that communicate with each other and with the environment via signals.

1.3 How can one compare the two languages?

UML and SDL can be compared on the diagrams that they offer. The main diagrams that will be considered in this report are:

1. Class Diagrams in UML and Block Diagrams in SDL
2. State Charts in UML and Process/Procedure Diagrams in SDL
3. Sequence Diagrams in UML and Message Sequence Charts used in conjunction with SDL

After a basic discussion of the above diagrams, the report examines the general similarities and differences between the two languages. Lastly, it discusses the future of the languages.

2 UML Class Diagrams and SDL Block Diagrams

UML class and object diagrams describe the static structure of a system. In SDL, the static structure of a system is described in system and block diagrams.

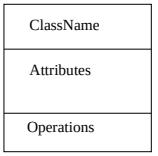

Figure 1: Class Symbol

Figure 2: Interface Symbol

2.1 Class And Object Diagrams

A *class diagram* contains classes, interfaces, collaborations and relationships.

A *class* is a set of objects with the same attributes, operations, relationships and semantics. Classes may implement one or more interfaces. The notation for a class is a rectangle as shown in figure 1.

An *interface* is a collection of operations that specify a service which a class should provide. It describes the externally visible behaviour of an element. It only defines operation specifications and not operation implementations. The notation for an interface is a circle, which is usually attached to the class that realises it. This is shown in figure 2.

A *collaboration* is an interaction and defines a group of elements that work together to provide some cooperative behaviour. Collaborations have structural and behavioural properties. A class may participate in more than one collaboration . A collaboration is shown as an ellipsis with dashed lines as depicted in figure 3.

A *dependency* relationship is a semantic relationship between two objects. It is used to depict the fact that a change to one object may affect the semantics of another object. In other words, it represents that one object is dependent on another [1]. It is shown as a dashed line, which may be directed. This is shown in figure 4.

An *association* relationship is a structural relationship that describes a set of links. A link is a connection between objects. A special type of link is aggregation, which shows a structural relationship between a whole, and its parts. An association is depicted as a solid line [1]. It may have adornments such as multiplicity and role names such as in the example in figure 5.

A *generalization* relationship shows the relationship between a general element (or parent) and a specialised version of this element (or child)[1]. It is

Figure 3: Collaboration Symbol

Figure 4: Dependency Symbol

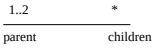

Figure 5: Association Symbol

depicted as solid line with a hollow arrowhead pointing toward the parent. This is shown in figure 6.

A *realization* relationship is a relationship between two elements in which one element specifies a contract that the other element carries out. These occur between interfaces and classes or components that implement them as well as between use cases and the collaborations that carry them out [1]. A realization is shown as a directed arrow with a filled in arrowhead and can be seen in figure 7.

Other relationships include refinement, trace, include and extend which will not be discussed here.

Class diagrams depict the static view of a system and are the most common type of diagram used in object-oriented systems.

An *object diagram* contains a set of objects and their relationships. Object diagrams show instances of things found in class diagrams. They also show the static view of a system.

2.2 System and Block Diagrams

An SDL system consists of a number of blocks. SDL Blocks represent components of a system. Each block may be connected to other blocks or to the boundary of the system via channels. Channels convey signals between blocks. A block contains a number of process descriptions and possibly subblock descriptions [2].

A *system diagram* gives a high level view of a system. The system contains everything to be specified. A system diagram contains:

- system name
- signal descriptions - for the signals exchanged between blocks in the system or between blocks and the environment.
- channel descriptions - for the channels that interconnect blocks in the system and that connect blocks to the environment. A channel description contains the name of the channel, a list of signals that can be transported by that channel and the identification of the end points of that channel. A channel is depicted as a line with an arrowhead as shown in figure 8.

Figure 6: Generalization Symbol

Figure 7: Realization Symbol

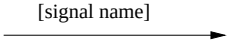

Figure 8: Channel Symbol

- data type descriptions - for user defined data types that exist in the system an its environment.
- block descriptions - for the blocks that make up the system. A block is shown as a rectangle with cut off edges as depicted in figure 9.

System diagrams may also contain signal list descriptions and macros descriptions, which are not discussed here [2]. Descriptions that are written in the textual syntax of SDL are placed within a text symbol as shown in figure 10.

A *block diagram* gives more detail than a system diagram [2] . A block diagram contains:

- block name
- signal descriptions - for signals within that block.
- signal route descriptions - for the signal routes that connect the processes within a block together as well as to the environment of the block. A signal route is depicted as a line with arrowheads at the ends of the signal route symbol. This is shown in figure 11.
- channel to route connections - which describe the connections of the channels outside the block with the signal routes within the block.
- process descriptions - for the process types that describe the behaviour of the block. A process is shown as a rectangle with cut off edges as shown in figure 12.

The above diagrams show the static view of the system and can be considered equivalent of UML class and object diagrams.

Figure 9: Block Symbol

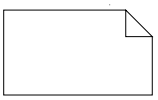

Figure 10: Text Symbol

Figure 11: Signal Route Symbol

3 State Charts in UML and Process/Procedure Diagrams in SDL

3.1 State Charts

UML state charts show how an object moves from one state to another and the rules that govern the change of state. State charts provide a state overview and focus on states. State charts usually have a start and an end condition.

A *statechart diagram* contains a state machine with states, transitions, events and activities. It depicts a dynamic view of a system and is used to model the behaviour of a class, interface or a collaboration. It can also be used to show the event ordered behaviour of an object.

A *state machine* represents a sequence of states an object or an interaction undergoes in response to certain events. It is used to represent the behaviour of collaboration of classes or of a specific class. It is made up of states; transitions which are a flow from one state to the next; events which are the things that trigger transitions and activities which are the response to transitions [1]. A state is shown as a rounded rectangle. This is shown in figure 13.

3.2 Process and Procedure Diagrams

SDL process diagrams show state-to-state behaviour as well. However, they focus on transitions, actions and the control of flow.

An SDL process is an extended finite state machine. It works independently or concurrently with other processes. Processes can cooperate by sending asynchronous signals to one another. A process can also communicate with the environment of the system via signals. The behaviour of a process changes as it responds to external stimuli and is represented by a number of states and transitions. A process can use and manipulate data stored in variables local to its finite state machine [2].

Figure 12: Process Symbol

Figure 13: State Symbol

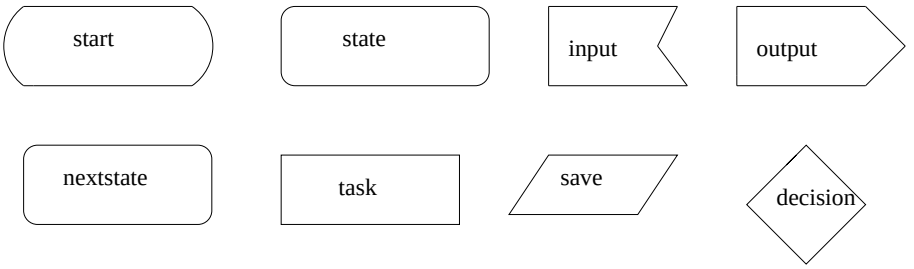

Figure 14: Basic Symbols in a Process Diagram

An SDL procedure is a finite state machine within a process. It is created when a procedure call is interpreted and dies when it terminates. SDL procedure diagrams help make process diagrams less cluttered by moving the detail to a separate diagram.

A *process diagram* describes a process [2]. It contains:

- process name
- formal parameters
- variable descriptions
- timer descriptions
- procedure descriptions
- descriptions of the finite state machine of the process

The basic symbols used in a process diagram are shown in figure 14.

A *procedure diagram* is similar to a process diagram. Two symbols that are differ in a procedure diagram are the start symbol and the return symbol as depicted in figure 15.

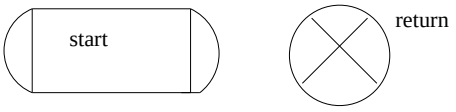

Figure 15: Symbols in a Procedure Diagram

3.3 Differences in the diagrams

Looking at statecharts in UML and finite state machines (in process or procedure diagrams) in SDL, one notices differences between the two. UML statecharts focus on states and give a much better overview of the different states in which an object can be. Furthermore, there is a lack of good notation in UML to model transitions in greater detail [4].

SDL finite state machines, conversely, focus on transitions and are much more detailed than statecharts. Several symbol differences in the diagrams exist as well. For instance, UML include the notion of nested hierarchical states and entry as well as exit level actions. This is not done in SDL except as flattened states with entry and exit level actions modelled in appropriate transitions. Another difference is that a procedure in SDL is represented by an operation or a state machine in UML.

A further dissimilarity is that input in SDL process/procedure diagrams are represented as transition triggers in state charts. Saved signals in SDL are represented by deferred events in UML and nested states in UML are modelled as flattened states in SDL. Lastly, a dash state in an SDL finite state machine is shown as an internal transition in an UML state chart. The notation differences are shown in figure 16.

4 Sequence Charts in UML and Message Sequence Charts used in conjunction with SDL

4.1 Differences in the diagrams

UML sequence diagrams are used to model the logic of usage scenarios. A usage scenario describes a potential way that your system is used. Thus a sequence diagram shows a pass through part of or a whole use case. A sequence diagram emphasises the time ordering of messages.

Similarly, MSCs depict a path taken through an SDL specification. According to [5], they show sequences of signals sent between two or more processes. Therefore they are used to help understand the information that needs to be passed between parts of a system. Furthermore, they identify the logical time order in which information is available and can be processed. A MSC entity name should correspond to the name of an equivalent entity in the associated SDL specification.

There are several differences between UML sequence diagrams and MSCs. Firstly, MSCs have inline expressions. These provide a compact notation to represent minor variations in an MSC. The variations could be alternative paths, optional parts, repetitions or exceptions. Within a sequence diagram in UML, it is more difficult to define variability in a single diagram.

Secondly, in MSCs one can include references to other MSCs within a single diagram. This allows one to leave out certain details in one diagram and put them in a separate diagram for easier readability. In UML, on the other hand, one cannot reference one sequence diagram from another.

Thirdly, in MSCs, high-level MSCs show how different diagrams are related to each other. This is done in a similar fashion to an inheritence hierarchy. These high level charts provide a good overview and a compact way of describing

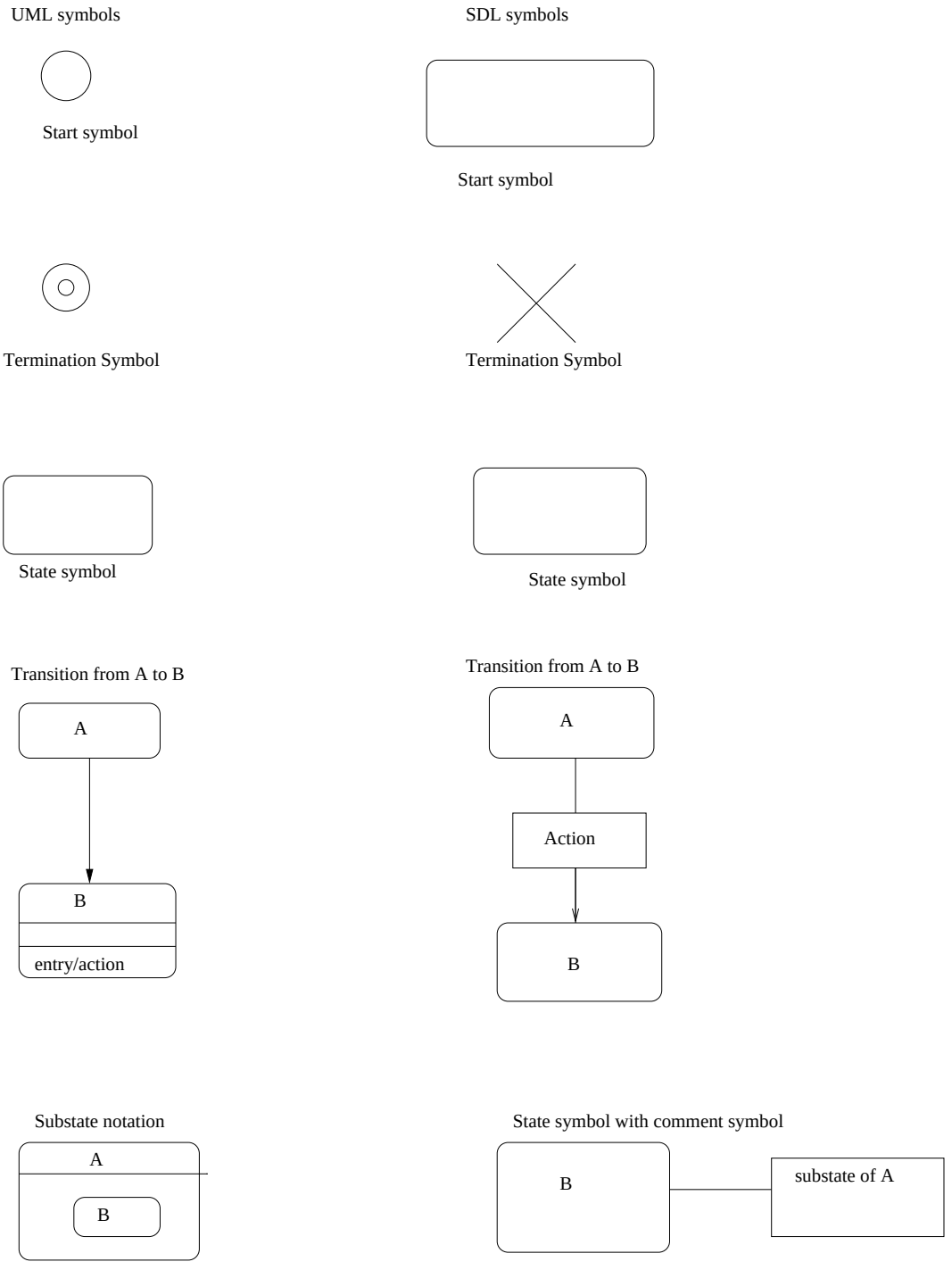

Figure 16: UML and SDL notation differences in state charts and process diagrams respectively - UML symbols are shown on the left and SDL symbols are shown on the right

several MSCs. This feature of MSC is useful for large requirements specifications composed of many MSCs because it helps to structure the diagrams. In UML, showing an overview of how the sequence diagrams are related is not possible.

Conversely, UML has guard conditions, transition names and the ability to express constraints, which one cannot do in MSCs. Thus the diagrams are similar but each has several different features from the other. The notation of the diagrams is also slightly different and is shown in figure 17.

4.2 Sequence Diagram Notation

In a UML sequence diagram, the rectangles at the top of the diagram represent classifiers or their instances. These may be use cases, objects, classes or actors. Objects are named in the following way: ObjectName: ClassName Object names are optional. If an object has no name, it is known as an anonymous object. Classes and actors are named in the following ways: ClassName Actorname These names are placed inside the rectangles representing the entities at the top of a UML sequence diagram. Dashed lines that run vertically from the rectangular entities at the top of a sequence diagram represent object lifelines. A lifeline depicts the life span of an object during the scenario being modelled. Long thin vertical rectangles on lifelines represent method invocations. Method invocations indicate processing that is being performed by the target object or class to fulfil a message. A cross at the bottom of a lifeline indicates object termination. This means that the object is destroyed and is removed from memory. Messages in sequence diagrams are represented by labelled arrows. These are the basic symbols used in UML sequence diagrams and can be viewed in figure 17.

4.3 MSC Notation

In a MSC, the notation for a message is also a horizontal labelled arrow from one entity axis to another. An entity axis is the solid vertical line representing the lifeline of the entity. Each entity is also depicted as a rectangle as in UML. However, the labelling conventions differ slightly. In MSCs, the instance kind is placed inside the rectangle representing the entity and the instance name is placed above it. Another difference between MSC notation and UML notation is that every instance axis in MSC terminates with a filled in box. MSCs have no formal notation for actions. An action taken is defined by text in an action box. This has no meaning in MSC but can be given formal meaning in SDL. This is the basic notation for MSCs.

5 What are the similarities between the two languages?

SDL and UML have many things in common - both languages are object oriented and have a graphical representation. Good readability is a feature of both languages and extensive tool support exists for SDL and UML. Lastly, both languages can be used to model many different types of software systems.

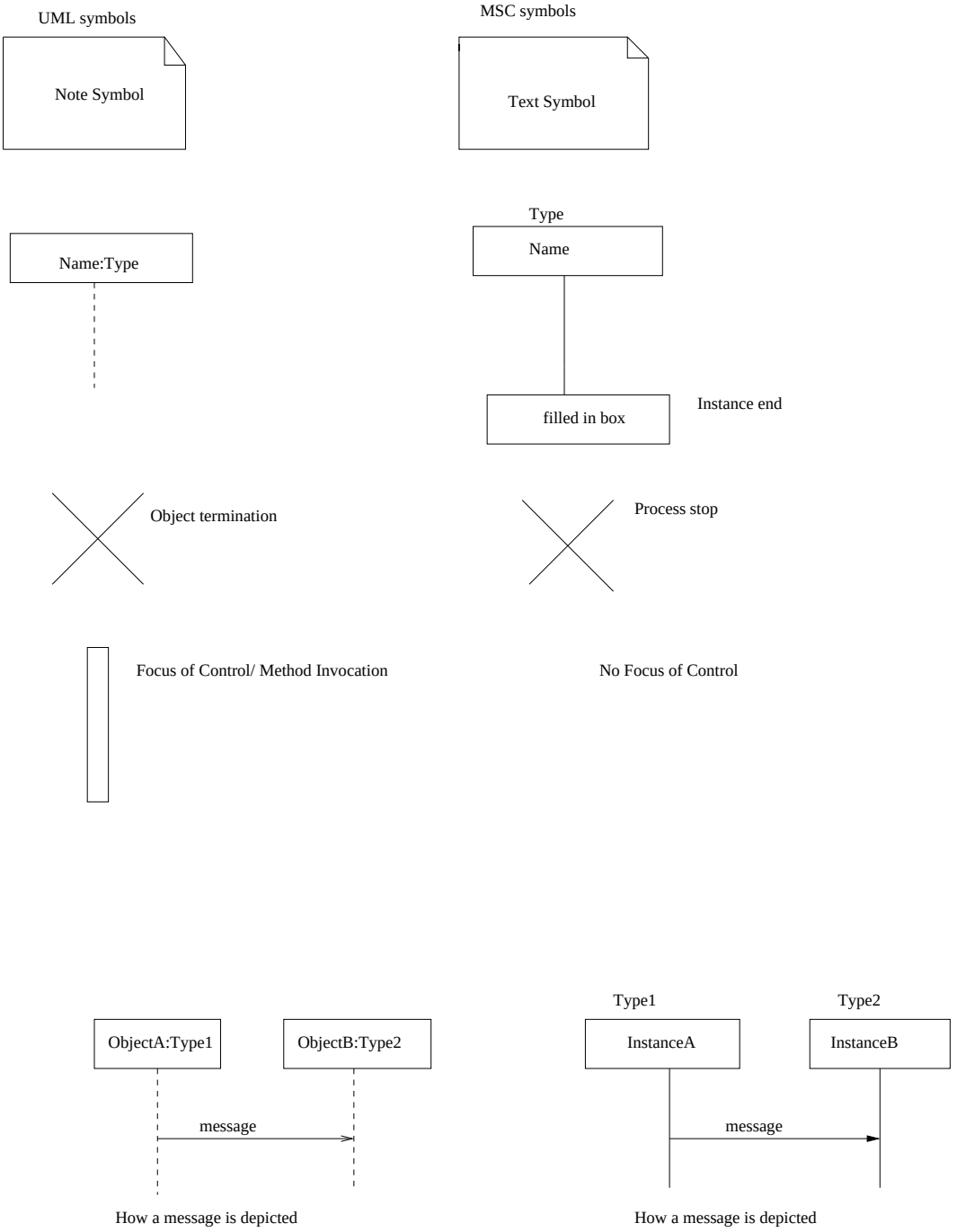

Figure 17: UML and MSC notation differences in sequence diagrams and message sequence charts - UML symbols are shown on the left and MSC symbols are shown on the right

6 What are the differences between the two languages?

However, many distinctions between the two languages are present. This section compares and contrasts SDL and UML.

Firstly, SDL has both a graphical (SDL/GR) and a textual syntax (SDL/PR) [3]. This makes SDL a precise and complete language, which is unambiguous and has formal semantics.

The formal semantics of SDL support the automatic generation of code from SDL specifications. This gives one the ability to verify and validate specifications written in SDL to ensure that they conform to their requirements. SDL also allows one to generate test cases to check specifications for correctness.

It is for this reason that SDL is widely used in the telecommunications industry. When specifying protocols, for example, ensuring that the protocol runs correctly is imperative. SDL can, of course, be used to model many other types of system as well.

UML, on the other hand, is just a graphical language. All textual parts of a UML model are written in English prose and are usually placed in adornments on a UML diagram or in separate documents.

The only textual language that is a part of UML is Object Constraint Language (OCL)[7]. It is used to specify constraints over entities in a model. Constraints expressed in English prose may be unclear and OCL eliminates these ambiguities due to its formal nature. However, at present, OCL is not regarded as the textual format of UML models. Thus UML can be regarded as more flexible and expressive than SDL due to fewer formalities.

Also, UML's lack of notational and semantic detail often makes it favoured as a formal specification method because it is easier for users to grasp. Yet, it also means that UML has no formal semantics which is disadvantageous. This point is elaborated on below.

UML is used for modelling many different types of software systems. Amongst different specifications for different systems, the UML symbols and concepts may have different meanings. According to [8], the interpretation of a UML model also depends on the background and environment of the reader. Furthermore, due to the extensibility of UML via the use of stereotypes, constraints and tagged values, different UML documents may have entities unique to those documents.

For this reason, although UML has a standard notation, it does not have a standard meaning and this complicates the process of trying to generate code from, simulate or execute UML models. Therefore one is unable to check UML models for correctness and this puts UML at a great disadvantage as compared to SDL.

Another difference between SDL and UML is that UML has many more diagrams on offer than SDL. As a result, in UML, one can get many more views on the same information. For instance, one can get a functional view of the system (in use cases), a structural view (in class and object diagrams), a dynamic view (in collaboration diagrams and sequence diagrams) and an implementation view (in deployment and component diagrams) of a system.

Also, use case diagrams and collaboration diagrams are unique to UML and have no direct equivalent in SDL. Use case diagrams are useful because they

show how users of a system interact with the system. They easily demonstrate whether the system is doing what is required of it. Collaboration diagrams are convenient because they show one the structural organisation of objects that send and receive messages. This is one of the advantages of UML as compared with SDL.

Moreover, as [9] states, in UML, one can more easily see how classes are related by dependencies and associations, e.g. by inheritence/generalization hierarchies, at a glance. In SDL, these relations are not as easily apparent from just looking at the four basic SDL diagrams. For instance one cannot simply see how type definitions are related to each other via the textual format of this information as it is presented in SDL.

On the other hand, SDL diagrams form a much nicer hierarchy of detail from the high-level system diagram to the lowest level of detail in procedure diagrams [4] UML diagrams are not arranged in such a step-by-step decomposition of detail.

One major disadvantage of UML as mentioned in [4] is that if one specifies a model in UML, one has to manually implement it. At most, tools presently exist that can generate a skeleton code from a UML specification but one still has to fill in the gaps in code manually. SDL is superior in this regard as it supports automatic code generation. This also cuts down development time as one's specification gives one an application without having to code one line manually!

7 How can one translate between the two languages?

Many mappings exist between the two languages in spite of their differences. An example of such a mapping is given below. UML classes can be mapped onto SDL types. For instance, classes with the same attributes and behaviour can be mapped onto process types. Another example would be that of container classes mapping onto block types.

Some concepts in SDL have no direct or obvious equivalent in UML and vice versa. This section briefly discusses some of these constructs. In UML, there is no construct such as macros [10]. Macros definitions allow one to place a description in one part of a system and reference that description from any other part of the system [3]. Data modelling in SDL also cannot be mapped onto UML directly [10]. SDL models data in great detail whereas UML does not.

In addition, according to [9], the concept of interfaces in UML cannot be directly represented in SDL due to a lack of a corresponding construct. Furthermore, SDL does not have a construct equivalent of multiple inheritence in UML. In SDL, only single inheritence is supported [10].

Lastly, receiving priority in SDL cannot be modelled in UML directly [11]. Receiving priority refers to the fact that a receiving process in SDL can specify that a given input has priority over all other input. However, with the use of extra stereotypes, this concept can be modelled in UML [10]. These are just several of the concepts that differ between the two languages.

8 Conclusion

At present, UML and SDL are viewed as complementary languages. The two languages are increasingly being used together to specify software systems. UML is used in the analysis (of requirements) and early design parts of the software engineering process as it is more flexible and provides better overviews. More detailed design and behaviour specifications are done in SDL with the advantage of being able to test these for correctness.

Other ventures that have been undertaken using the two languages include attempts to show how one can be mapped to the other [10]. Selic and Rumbaugh [11] have written a paper showing how SDL can be mapped to UML and ITU have released a recommendation Z.109 showing how UML can be mapped to SDL.

[10] states that this ITU specification defines rules for translating UML class diagrams and statecharts into SDL equivalent architecture and behaviour. The translation defined in the specification gives the UML model SDL semantics. This allows for the compilation of UML models as well as simulation.

Furthermore, the latest version of SDL, SDL-2000 has tried to align SDL with UML [10] and includes descriptions of how to use UML models within SDL [4]. The Object Management Group (OMG) who are responsible for standardising UML are , in turn, trying to incorporate the use of SDL 2000 in the latest version of UML - UML 2.0.

Thus UML 2.0 will include an extension known as Action Semantics as well as an improved version of OCL. Action semantics [13] lets one express actions as UML objects and define actions that occur in state machines or class operations in more detail than previously. According to [4], actions semantics have been included to enable UML to specify behaviour in sufficient detail to allow for execution and verification of UML models.

Also, many companies such as Telelogic Tau are interested in the combined use of SDL and UML [12]. Recently, Telelogic released a tool that combines the languages, allows one to simulate specifications written in the combined language and generate test cases for specifications.

In conclusion, the future of both languages looks bright since they each have their supporters. Also, the developers of the respective languages are actively trying to remedy problems associated with both the methods. At this stage, it seems that UML and SDL are moving toward becoming a combined specification language that would be beneficial to all.

References

[1] Grady Booch, James Rumbaugh, Ivar Jacobson *The Unified Modelling Language User Guide*, Addison-Wesley, 1999.

[2] Ferenc Belina, Dieter Hogrefe, "The CCITT-Specification and Description Language SDL" , *Computer Networks and ISDN Systems 16 (1988/89) 311-341*, Elsevier Science Publishers B.V. (North-Holland), 1989.

[3] Ove Faergemand, Anders Olsen, "Introduction to SDL-92", *Computer Networks and ISDN Systems 26 (1994) 1143-1167*, Elsevier Science Publishers B.V.(North-Holland), 1989.

[4] Morgan Bjorkander , "Graphical Programming using UML and SDL". http://www.telelogic.com/download/paper/graphicalprogramming.pdf

[5] O Faergemand, A Olsen, B Moller-Pedersen, R Reed, J R W Smith, *System Engineering using SDL-92*, Elsevier Science Publishers B.V.(North-Holland), 1994.

[6] International Engineering Consortium online education, " SDL -specification and description language",http://www.iec.org/online/tutorials/sdl/.

[7] Object Constraint Language, http://www-3.ibm.com/software/ad/library/standards/ocl.html.

[8] Jos Warmer, " The future of UML" http://www.klasse.nl/ocl/index.html.

[9] E. Holz, "Application of UML in the SDL Design Process", SAM98 Workshop, Berlin, June 1998 http://citeseer.nj.nec.com/268645.html.

[10] Analysis to design - HOORA . http://www.hoora.org

[11] B Selic, J Rumbaugh, "Mapping SDL to UML", Rational Software white paper, 1999. http://www.rational.com/products/rosert/prodinfo/reading/sdl2umlv13.pdf

[12] "Telelogic First Tool Vendor To Provide Full Support For Both UML and SDL; Teleogic Tau Significantly Reduces Time To Market Large Scale Projects", Screaming Media, Business Wire, September 1999. http://industry.java.sun.com/javanews/stories/story2/0,1072,19107,00.htm

[13] Action Semantics. http://www.omg.org/gettingstarted

[14] SDL Forum Society http://www.sdl-forum.org/

[15] Rational Software www.rational.com