SDL Combined with UML

BIRGER MØLLER-PEDERSEN

The ITU-T Recommendation Z.109, “SDL combined with UML”, defines a one-to-one mapping between a subset of SDL and a specialised subset of UML. With this mapping it is possible to use UML for what SDL is good at (multiple views of the same system, informal object models, and property model views) and SDL for what SDL is good at (detailed and formalised object models, especially with respect to execution semantics).

1 Introduction

This is a presentation of the ITU-T Recommendation Z.109, “SDL combined with UML” [1], in a form that is assumed to be somewhat more readable than the Recommendation itself.

Z.109 defines a specialisation of a subset of UML [2] that has a one-to-one mapping to a subset of SDL. The semantics of this specialisation is given by the semantics of the corresponding SDL. This is also the case with the CORBA/IDL profile and any other language specific UML profile. The intention with Z.109 is however not to use the specialised UML instead of SDL, but to use SDL combined with UML. This means that even though Z.109 is based upon SDL for the semantics, not every concept in SDL has a mapping to UML. As they will be used in combination, there is no reason to make an artificial and complex mapping for concepts that are better supported by SDL and for which UML is not the right notation. For example, SDL has support for detailed specification of the object structure of systems and for detailed specification of behaviour, while UML is not meant to have this kind of support.

This presentation introduces a UML model of the most important concepts from the SDL subset. This model is not used to define the mapping in Z.109, but it may form the basis for the definition of a profile solely based upon the concepts of SDL, i.e. without requiring the language SDL as such. This UML model of SDL is intended for readers familiar with UML and plays the same role as the meta (UML) model that forms the basis for the definition of UML. Readers familiar with SDL may read this model as an alternative to the abstract syntax of SDL, but knowledge about the abstract syntax of SDL is not needed in order to read the rest of the document.

With the understanding of SDL in terms of the UML model, it should be possible for designers to make UML models without knowing the detailed mapping and the detailed semantics of SDL, and still use the UML in a way that lends itself to a mapping to SDL for detailed design.

2 Overview

UML/SDL Coverage

The main difference between SDL and UML is not that UML is especially well suited for analysis and SDL especially well suited for design. With support for associations, SDL may be used for making analysis object models in terms of classes and associations, as well as for design.

The main differences between UML and SDL are that

- UML is a collection of concepts and notations for several views of the same system: e.g. Object-, State Machine-, Use Case-, Collaboration and Interaction views;
- SDL is a language (with concepts, abstract grammar and graphical/textual grammars) focussing on the Object- and State Machine views of a system. For these views, SDL is however a complete language with static and dynamic semantics and with concrete syntax (graphical/textual) for the specification of actions. Users of SDL rely on other languages like MSC for specification of interactions between instances;
- UML has a weak semantics with many variation points, while SDL has a complete semantics, including execution semantics for state machines.

These differences are the reasons for Z.109. They are illustrated in Figure 1, which also introduces the following terms:

- UMLSDL: the specialised subset of UML with a mapping to SDL according to Z.109;
- SDLUML: the corresponding subset of SDL.

With the mapping defined in Z.109 it is possible for SDL users to use not only MSC for interaction modelling, but also to use UML for Use Case and Collaboration modelling. SDL users may also use the Object and Statemachine mod-
elling of UML at stages where the detailed semantics is not determined, and then turn to SDL when detailed specification is needed. With Z.109 it is also possible for UML users to use SDL for more precise models, including the specification of actions.

Z.109 implies no sequence in the use of UML and SDL. As indicated in Figure 1 a tool supporting Z.109 should be able to provide both the SDL- and the UML view of the subsets covered by Z.109.

Z.109 provides a mapping between the UML meta-model and the (abstract) grammar of SDL. For the notation in UMLSDL, the notation defined in SDL (Z.100) can be used, where this is appropriate. Otherwise the UML Notation Guide applies. One example of a difference between the notation defined by SDL and the UML Notation Guide is the notation for tagged values. While UML, and thereby UMLSDL, have tagged values enclosed by {}, SDL uses keywords preceding the type names.

Presentation Structure

The rest of this presentation is a description of the main concepts of SDL and their representation in UML according to Z.109. It is based upon an example, an Automatic Teller Machine (ATM), in order to illustrate the use of Z.109 for the combined use of SDL and UML. The example is only intended to give an idea on how the mapping between UML and SDL may work and does not claim to cover all details of the mapping.

The presentation requires a detailed knowledge of UML – on the other hand there is no reason to use Z.109 without a fairly good knowledge of UML. The whole idea of Z.109 is to enable users of SDL or users of UML to take advantage of the combined use of UML and SDL.

For each of the main SDL concepts the following is described:

- A short textual description that describes the SDL concept;
"Definition of a profile:

- In general, a UML profile is a mutually consistent set of predefined specifications that collectively customize UML for a specific domain or purpose (e.g. a "unified process" profile). The specifications in a profile typically consist of UML stereotypes, tagged values, constraints, notational elements (icons, diagrams, etc.), and other possible specifications. By definition, a profile does not extend UML by adding any new basic concepts and fully conforms to the semantics of the general UML standard.

- More precisely, a UML profile is defined as a specification that does one or more of the following:
  - Identifies a subset of the UML meta-model (which may be the entire UML meta-model);
  - Specifies “well-formedness rules” beyond those specified by the identified subset of the UML meta-model. “Well-formedness rule” is a term used in the normative UML meta-model specification (ad/97-08-04) to describe a set of constraints written in UML’s Object Constraint Language (OCL) that contributes to the definition of a meta-model element;
  - Specifies “standard elements” beyond those specified by the identified subset of the UML meta-model. “Standard element” is a term used in the UML meta-model specification to describe a standard instance of a UML stereotype, tagged value or constraint;
  - Specifies semantics, expressed in natural language or in any appropriate language, beyond those specified by the identified subset of the UML meta-model;
  - Specifies common model elements (i.e. instances of UML constructs), expressed in terms of the profile.

- The above definition is taken verbatim from the UML 1.3 specification with one important exception: the ability to extend the UML meta-model with new metatypes. The latter capability may be included in a more comprehensive future definition of a profile, but is out of the scope of the current profile."

Z.109 follows this definition, with the exception of the definition of the semantics. This is not described as part of the text, but obtained by the mapping to the SDL subset.

3 Agents and Agent Types

An (analysis) object model of an ATM will typically include a class diagram with classes and associations and one or more collaboration diagrams showing the interaction between the involved objects for selected scenarios. In Figure 2 we have identified that an ATM consists of objects that model the panel, the validator and the cash dispenser, and we have specified the associations between the classes. The object model includes associations between ATM and User and between ATM and CentralUnit because this is the only way to specify that they do not only have associations to parts of the ATM, but also to the ATM as such.

For the purpose of this presentation, we have only included one collaboration diagram and just used the structural part of the collaboration to specify which instances will be linked (Figure 3). A full collaboration diagram may include the specification of the interactions between the objects. Note that the ATM instance is not present in the collaboration diagram. It could be, but it would just be an object similar to the other objects: it is not possible to specify as part of the collaboration diagram that the ATM object is composed of the Panel-, Validator-, and Cash-Dispenser objects.

Given the UML Object model in Figures 2 and 3, there is no unique SDL specification. As soon as the UML is elaborated to conform to UMLSDL, the UML model is a partial specification of a potentially more detailed SDL specification. In order to elaborate the UML model, we have to know what the concepts of UMLSDL are and how they are expressed in the specialised UML.
**SDL: Agents**

An SDL System consists of Agents that are connected by means of Channels. Agents may communicate by sending Signals or by requesting other Agents to perform Procedures.

An Agent may have both a StateMachine and an internal structure of Agents (a composite Agent). The internal Agents and the StateMachine are connected by Channels. The connection points for Channels are Gates.

**Conceptual UML Model.** Figure 4 gives a UML conceptual model of this part of SDL. An SDL specification may specify both singular Agents and types of Agents. Types correspond to classes in UML. Because UML only prescribes classes of objects (and not singular objects), Figure 4 only gives the UML model of Agent types. UML supports the notion of object diagram, but that is only for describing snapshots of UML run time objects and not for the prescriptions of object structures in the body of types.

Agents come in different kinds with different execution semantics: Block Agents are concurrent Agents with possibly interleaved execution of the transitions of the state machines, while Process Agents are alternating Agents with run-to-completion execution of transitions. The overall system is a special System Block Agent.

**UML Mapping.** An Agent type maps to a Class of active Objects, with constraints as described below. System, Block and Process agent types are mapped to classes with stereotypes “system”, “block” and “process”, respectively.

Part of the structural content of a composite Agent (in terms of Agents connected by means of Channels) is mapped to a combination of Composition and the structural part of a Collaboration. The representedClassifier of the Collaboration is the Class representing the composite Agent type. Channels between sub-agents are represented by AssociationRoles in the Collaboration. The ClassifierRoles of the Collaboration represent the types of the sub-agents. An eventual stateMachine of the composite Agent is not (or rather cannot be) represented as part of the Collaboration.

Types defined locally to an Agent type are mapped to Classes in the Namespace of the Class representing the enclosing Agent type.

**Base Class for stereotypes “system”, “block” and “process”**

- Class for Agent type without internal Agents

---

**Figure 3 Structural part of a Collaboration Diagram**

**Figure 4 A UML conceptual model of the basic SDL modelling concepts and relationships**

**Figure 5 Kinds of Agents**
With this introduction to the notion of agents in SDL and how they are mapped to UML SDL, we can now elaborate the general UML model of the ATM. We decide that all involved classes shall be classes of block agents, because they execute concurrently with each other. This is reflected in Figure 6, where the classes have been stereotyped with “block”.

A class diagram as in Figure 6 implicitly defines a set of classes in a package (or in the namespace of another class). This is according to the definition of UML, and a UML tool will normally provide this information in a browser. The corresponding SDL specification, see Figure 7, includes the information of the enclosing package graphically. SDL has a package diagram that contains class symbols for the types that are defined in that package. As associations are supported by SDL and as class symbols represent type definitions, there is a one-to-one mapping between the two specifications.

Icons can be used as an alternative to stereotypes, see Figure 8. The icons for the different kinds of agents are defined in Z.100.

In addition to the package diagram with type references and associations, the SDL specification includes a type diagram for each of the types, see Figure 9. The package diagram with the class symbols only tells that there is a number of types defined in the scope of the package, while the detailed specifications of the types are given in the separate type diagrams.

As demonstrated so far, UML can be used in situations where the object kinds have not been decided yet. Figure 2 only specifies that some kind of ATM objects will be composed of objects of other classes, but not whether these are concurrent objects or not.

As illustrated in Figure 7 and Figure 8, a class symbol represents a type. If the name of the class has no qualifier, then the class symbol specifies that a type is defined in the scope defined by the enclosing diagram, and that the complete type definition is given in a (referenced) separate, complete type diagram, as those found in Figure 9. If the name contains a qualifier (a path expression denoting some other scope), then the class symbol just represents an application of a type. The type is then defined in the scope unit denoted by the qualifier.

A class symbol is, however, not just an indication that there is a type being defined (in a separate diagram), but also a partial type specification. Specification of an attribute in a class symbol, see Figure 10, implies a corresponding speci-
ification of a variable in the corresponding SDL type diagram as in Figure 11.

4 Associations
The example includes both ordinary associations and compositions. This presentation does not make any effort to convey the details on the use of the many properties that associations can have.

SDL – Associations
Types can be associated. Associations are defined at the level of the types being involved. Associations are properties of the enclosing entity. Associations have no implied semantics for the types involved.

UML Mapping. The notion of associations in SDL is a strict subset of associations in UML, so the mapping between SDL and UML is straightforward. As we shall see below, associations are also used to represent two special concepts of SDL:

• Internal structure of Agents, represented by a combination of Composition and Collaboration in UML-SLD;
• Gate with endpoint constraint, i.e. a gate that can only be connected with instances of a certain type, represented by a stereotyped Association.

5 Internal Structure of Agents
Assume that the next step is to model the details of block type ATM. In Figure 2, Figure 6 and Figure 7 it is only specified that instances of classes may be linked (as the classes are associated), and it is specified that ATM instances will contain instances of Panel, CashDispenser and Validator (by composition). Figure 3 says that the instances of Panel, CashDispenser and Validator collaborate, and it also tells which of these collaborate with the user and with the central unit.

The composition in Figure 6 can be mapped to the corresponding composition in SDL (Figure 7 and Figure 8), but it can also be mapped to a partial specification of the more detailed internal structure of agents. This is done in the SDL diagram for block agent type ATM in Figure 12. It specifies that each ATM instance will contain a number of sub-agents that are connected by means of channels.

This detailed SDL specification introduces two new concepts: Interfaces and Gates. The following defines these and provides the mapping to UML.

Figures 9, 10, and 11... and the corresponding block type diagrams, here only partially specified.

SDL – Interfaces and Gates
An Agent may have a number of required and implemented Interfaces. An Interface defines Signals, Variables, RemoteProcedures and Exceptions. A Signal defines the types of the data (parameters) that will be sent with each signal instance. A RemoteProcedure defines the signature of procedures that may be exported by Agents and thereby be requested by other Agents.

Interfaces are associated with Gates. Gates are connection points for Channels connecting Agents. Communication between Agents takes place via Channels.

An implemented Interface defines which Signals and which RemoteProcedure call requests that may be sent to an Agent. A required Interface defines which Signals the Agent may send and which RemoteProcedures it may request from other Agents.

Figure 9... and the corresponding block type diagrams, here only partially specified.

Figure 10 Partial type specification...

Figure 11... corresponding to a textual specification in the diagram.

Figure 12 SDL specification of the ATM block type.
A Gate with an EndpointConstraint can only be connected to Agents of the same type as or to a subtype of the Agent type of the constraint.

**UML Mapping.** An Interface is mapped to an Interface. UML Interfaces can only have Operations (corresponding to procedures), so Signals, Variables and Exceptions are mapped to stereotyped Operations.

It is not possible to represent the Gates of an Agent in UML directly according to the conceptual model, unless representing them as attributes or objects. A Gate is, however, not an instance but rather a connection point and as such similar to an Association of UML Objects. Therefore, Gates are mapped by means of different kinds of Associations:

- For gates as connection points between instances of two agent types the mapping is to an association between the two types.
- For a gate of a single agent type the mapping is to a stereotyped association. The name of the association maps to the name of the gate. The type at the other end of the association is either the type of an endpoint constraint, or no type in case there is no endpoint constraint.

**Base Class for gate**
- AssociationEnd for the definition of a Gate as a possible connection point based upon associations between classes representing Agent types;
- Association stereotyped with gate for the definition of a gate (possibly with endpoint constraint) as part of the definition of an Agent type.

The example in Figure 15 shows the use of stereotype “signal” to specify that the interface PanelValIF defines a set of signals. The corresponding SDL graphical interface definition is also given.

If we were to map Figure 12 back to UML, the gates of the ATM block agent type are not specified as objects, even though the conceptual model in Figure 4 defines gates as objects. The gates of the ATM are defined by means of a combination of interfaces and associations to other classes. In Figure 16 one of the gates of the ATM block type is represented by two interfaces, one for incoming signals and one for outgoing signals.

In Figure 17 the gates of two of the agent types are defined by the role names of an association between the two types. The implemented and/or required interfaces are specified either on the
types (as in Figure 17) by means of use and realises dependencies, or as part of the role names (not shown here). If more than one gate at each agent type should be specified, then the role name alternative must be used in order to specify which interfaces belong to which gates.

Figure 17 illustrates that gates with two-way interfaces are better specified in SDL. This is not surprising, since this is one of the distinguished features of SDL. Z.109 defines the mapping, but in actual use it is recommended to specify the gates as part of the SDL specification.

From the SDL specification in Figure 12, it is possible to apply the mapping from SDL-UML to UML-SDL. This is illustrated in Figure 18 and Figure 19. The mapping is to a combination of a Composition with representation of the gates of the composite class and a Collaboration with roles representing the parts of the composite that interacts.

The internal connections are mapped to associations between the classes (Figure 18) and to association roles between the roles of a Collaboration (Figure 19). Note that compared with the initial collaboration (Figure 3), the collaboration in Figure 19 does not include the User and the CentralUnit roles.

The association roles linking P with V and V with CD correspond to the channels of the SDL model. For the purpose of specifying the contents (in terms of objects) of a type of objects, (here the type ATM), the SDL diagram is superior. The UML specification provides the specification at the class composition level and a separate collaboration diagram, while the SDL specification combines these two.

6 Signals
As described above, possible interactions between Agents by means of signals is defined by interfaces and gates. The signals themselves are defined by signal definitions, either as part of interface definitions or as part of packages or agents.

SDL: Signals and Signal Types
A Signal may carry a set of values called Parameters.

A Signal definition defines a type of Signal instances. An Agent sending a Signal does this by generating a Signal instance according to a Signal type and providing the actual Parameters. The receiving Agent may assign the values of the Parameters to local Variables.

Signals can be defined either in the Agent enclosing the Agents that use the signals for com-

Figure 17 Gates by means of Associations

Figure 18 Analysis model in UMLSDL, with only one of the gates of ATM represented in UML

Figure 19 Collaboration representing ATM
A Signal can be defined as a subtype of another Signal. The subtype inherits the Parameters of the supertype and may add Parameters. Inherited Parameters cannot be changed.

A Signal type defined in an Agent type can be defined to be a virtual Signal type. A virtual Signal type can be redefined/finalized by extension in subtypes of the Agent type with the virtual Signal type definition.

**UML Mapping.** Signal is in UML represented by the standard UML subclass of Classifier stereotyped with “signal”. The Parameters are represented by Attributes.

**Tags**
- Virtual - the Signal type is a virtual type
- Redefined - the Signal type is a redefined type, but still virtual
- Finalized - the Signal type is a finalized type, and no longer virtual

**(Stereotype) Constraints**
- A Signal type Classifier can have at most one super (Signal) type Classifier and it must be stereotyped with “signal”;
- A virtual, redefined and finalized Signal type Classifier must be defined in the namespace of an Agent type Classifier;
- Visibility of attributes representing Parameters does not apply.

A signal can in SDL be defined in a textual form, and in a combination of graphical and textual forms. As an example, the signal AcceptCard introduced in Figure 15 is defined in a Package called ATMsignals. Figure 21 provides the SDL and the corresponding UML specification.

**7 Specialisation of (Agent) Types, and Virtual Types**

In order to illustrate the use of tagged values, suppose that the panel of the ATM needs to be redefined for different kinds of ATM. The different kinds of ATM are defined by specialisations of the general ATM class. Both SDL and UML cover specialisation, although in a slightly different form.

**SDL: Agent Types and Subtypes**

An Agent type can be the specialisation of another Agent type. A specialisation may add properties to those specified for the supertype, including subAgents and Channels, and it may redefine/finalize virtual types and/or Procedures being defined in the supertype.

A virtual type/Procedure can be either redefined (in which case it is still virtual) or finalized (in which case it is no longer virtual). Redefinitions/finalizations must obey the constraint of the virtual. A virtuality constraint is in terms of a general type/procedure, and the redefinitions/finalizations must be subtypes/subprocedures of the constraint.

**UML Mapping.** Specialisation maps to Generalisation, with the constraint that there is only one superclass and that the kind of the superclass is the same as the kind of the subclass.

**Tags**
- Virtual - the agent type is a virtual type;
- Redefined - the agent type is a redefined type, but still virtual;
- Finalized - the agent type is a finalized type, and no longer virtual.

The virtuality constraint is mapped to the constraint association of the Class.

**(Stereotype) Constraints**
- An agent type Class can have at most one super (agent) type Class and it must be of the same kind as the agent type Class;
- A virtual, redefined and finalized agent type Class must be defined in the namespace of another agent type Class;
• A **Collaboration** representing the structural content of a super agent type **Class** is inherited by subtype **Classes**.

In SDL a general ATM type is specified by having the type of the panel (ATMpanel) defined locally to the type ATM and by having it defined as a virtual type, see Figure 22.

Note that the locally defined type ATMpanel is indicated by a type symbol within the diagram of the enclosing type. The full specification of the type ATMpanel will be in a separate diagram, see Figure 23. The heading of this diagram will give additional specification of the type, e.g. that it is constrained by the general type Panel, so that it can only be redefined to a subtype of Panel.

The constraint type Panel, see Figure 24, defines the common properties of panels, including the interfaces in terms of gates. By specifying the Panel as the constraint of the virtual type ATMpanel, it is enforced that all redefinitions have this interface.

In UML SDL, the fact that ATMpanel is defined in the namespace of ATM is specified by the plus encircled line, and the virtuality is specified by the tagged value “virtual”, see Figure 25. So far, we have only used the filled diamond type of UML graphics for Composition. Figure 25 includes the symbol enclosing style, in order to illustrate that a class symbol within another class symbol is not the same as name space containment. In both variants in Figure 25 the plus encircled line must be there in order to specify that the class is defined locally to the ATM.

The constraint of the virtual type is not shown in the class symbol for the virtual type.
8 State Machines

In order to illustrate state machines, the block type ATM is changed by substituting the Validator block with a state machine. The effect would be the same if the Validator block had its own state machine: the ATM state machine will execute concurrently with the state machines of Panel and CashDispenser.

The state machine of the ATM illustrates the notion of composite state. Figure 26 is a UML statechart diagram associated with the class ATM. In UML, the association between the class and statechart is part of the UML metamodel, but there is no graphical notation for it. In a Class Diagram there will be a class symbol for ATM, while a separate Statechart Diagram defines the state machine. There is nothing in the class symbol saying that this class has an associated state machine.

A StateAggregation is mapped to a CompositeState with isConcurrent=true. A StatePartition is mapped to a State with isRegion=true, but with a different semantics than UML (see above). isRegion is an attribute of the UML metamodel element CompositeState.

The SDL specification corresponding to Figure 26 has a block type diagram for ATM. This diagram shows the contents of ATM objects, including the fact that there is a state machine. The state machine is defined in a separate diagram.

The state symbol (with the name ATM) in the block type diagram in Figure 28 specifies that the ATM has a state machine. The definition of the state machine is given in a separate state diagram, see Figure 29.

Note that the state machine of ATM is defined by two state diagrams. The reason is that the state machine of ATM contains a composite state ReadAmount. In the state diagram ATM this state is specified by a state symbol, and its internal specification (in terms of states and transitions) is given in a separate ReadAmount state diagram. Even though there is a mapping between UML and SDL state machines at the meta-model/grammar level, the graphical syntax for state machines is very different. While SDL-2000 has (independently of Z.109) introduced UML-like class symbols for types and associations in the style of UML, the state machine notation of SDL is so different from the Statechart notation that composite states was introduced in SDL-2000 as part of the existing SDL notation. Z.109 can be used to switch between the SDL notation and the Statechart notation.

As demonstrated by the example above, the main rationale for Z.109 to cover state machines is that the view offered by UML Statecharts is

SDL: State Machines

States may be either BasicStates or CompositeStates. The StateMachine of an Agent is a CompositeState. A CompositeState has a number of States and Transitions. Transitions are triggered by events like input of a Signal or a remote Procedure call. Composite states may be StateAggregations, i.e. states with a number of StatePartitions. A StateAggregation is in one of the states in each StatePartition, while an ordinary CompositeState is in just one of its states.

UML Mapping. In the UML meta-model there is an aggregation between ModelElement and Statemachine, with role names context (the ModelElement) and behaviour (the Statemachine). The stateMachine of an Agent is represented by a Statemachine with the context associating it to the class of the Agent type.

The SDL specification corresponding to Figure 26 has a block type diagram for ATM. This diagram shows the contents of ATM objects, including the fact that there is a state machine. The state machine is defined in a separate diagram.

The state symbol (with the name ATM) in the block type diagram in Figure 28 specifies that the ATM has a state machine. The definition of the state machine is given in a separate state diagram, see Figure 29.

Note that the state machine of ATM is defined by two state diagrams. The reason is that the state machine of ATM contains a composite state ReadAmount. In the state diagram ATM this state is specified by a state symbol, and its internal specification (in terms of states and transitions) is given in a separate ReadAmount state diagram. Even though there is a mapping between UML and SDL state machines at the meta-model/grammar level, the graphical syntax for state machines is very different. While SDL-2000 has (independently of Z.109) introduced UML-like class symbols for types and associations in the style of UML, the state machine notation of SDL is so different from the Statechart notation that composite states was introduced in SDL-2000 as part of the existing SDL notation. Z.109 can be used to switch between the SDL notation and the Statechart notation.

As demonstrated by the example above, the main rationale for Z.109 to cover state machines is that the view offered by UML Statecharts is
good for the state overview, while the SDL view is good for specifying the details of transitions. In case of large state machines, the SDL way of dividing them into separate state diagrams is superior to the graphically nested form of state-charts.

9 Variables and Procedures of Agents

Somewhere behind the scene presented so far, there will be Account objects. Transactions on the ATM will imply transactions on the account of the user. A simple model of an account includes attributes like account number, balance and credit limit and operations like deposit, withdraw, open, and close.

The standard way of modelling this is to define an Account class. SDL and UML are similar here, with the exception that SDL makes a distinction between value- and object classes.

SDL: Object and Value Typed Variables

In addition to aStateMachine and sub-Agents, an Agent may have both attributes in terms of Object and Value type Variables, and operations in terms of Procedures. A Variable may be an exported variable, in which case other Agents may observe its value.
Variables and Procedures of Agents

UML Mapping. Variables are mapped to Attributes, and Procedures are mapped to a combination of Operations and Methods.

Base Classes
- Attribute for reference- and value variables;
- Operation/Method for Procedure;

Constraints
- Visibility = private is not applicable.

A local Variable is mapped to an Attribute with visibility = protected. An exported Variable is mapped to an Attribute with visibility = public.

Figure 31 illustrates the UML way of specifying a class with attributes and operations.

From this general UML model it is possible to elaborate into SDL in two different ways: either Account objects are active objects which execute their operations themselves, or Account objects are just data objects with associated operations.

In Figure 32 it has been decided to model the account by an active object, and therefore, the stereotype “process” has been applied.

In Figure 33 the corresponding SDL is given, in both graphical and textual form. The process type diagram is only sketched. In addition to the declaration of the variables and of the procedures, a process type diagram will typically contain the specification of the state machine, specifying in which states the different procedures will be accepted and executed.

The procedure symbols in the process type diagram in Figure 33 play the same role as class symbols for agent type. They specify that the enclosing type has a number of procedures and that the detailed specification of the procedures is to be found in separate procedure diagrams, just as for type diagrams, see Figure 34.

10 Procedures

Procedures have been introduced above as properties of Agents. The example will not contain any detailed definitions of procedures. Procedures are meant for specifying patterns of behaviour that an Agent can perform as part of transitions. Procedures can be specified in the same way as Agents: by means of a state
machine. Simple Procedures will just have one transition and no states.

**SDL: Procedure**

A Procedure is a pattern of behaviour specification that can be used by Agents performing the Procedure by calling it as part of transitions. Procedures may either be LocalProcedures or ExportedProcedures. A LocalProcedure is a procedure that is used locally in an Agent, while other Agents (according to the signature defined by a RemoteProcedure) may request the execution of an ExportedProcedure.

A Procedure can be a specialisation of another (general) Procedure, thereby inheriting parameters, eventual local variables and behaviour specification.

A virtual Procedure is a Procedure that can be redefined in subtypes of the enclosing Agent type. A virtual Procedure can be either redefined (in which it is still virtual) or finalized (in which case it is no longer virtual). Redefinitions/finalizations must obey the constraint of the virtual. A virtuality constraint is in terms of a general Procedure, and the redefinitions/finalizations must be subtypes/subprocedures of the constraint.

**UML Mapping.** Procedures are mapped to a combination of Operations, Methods and Classes. The signature of the Procedure is represented by an Operation, while the body is represented by a Method. A Procedure that is a specialisation of another Procedure is in addition represented by a Class (in the namespace of the Class representing the enclosing type) with stereotype “procedure” and with a Generalisation relationship to the Class (also stereotyped with “procedure”) representing the superprocedure.

**Base Classes**

- Operation/Method for Procedure;
- Class for Procedures that are specialisations of general Procedures.

**Tags**

- For Procedure:
  - Virtual - the Procedure is a virtual Procedure;
  - Redefined - the Procedure is a redefined Procedure;
  - Finalized - the Procedure type is a finalised Procedure.

**Constraints**

- LocalProcedures have private visibility only, while ExportedProcedures have public visibility only;
- Visibility = private is not applicable.

A LocalProcedure is mapped to an Operation with visibility = protected. An ExportedProcedure is mapped to an Operation with visibility = public.

**11 Data Types**

Data types are used to define the properties of attributes and parameters to signals and operations. The access code that follows each card, with a card identification number and a personal identification number, is an example of an attribute defined by a data type. The data type AccessCode will define a structure consisting of two fields: cardId and pin.

**SDL: Data Types**

SDL data types come in two different forms. A value data type <data type definition>:value:use in text defines a set of values. An object data type <data type definition>:object:use in text defines a set of objects.

Variables of Object types are references with an associated reference semantics. Assignment
between these variables is reference assignment, and two reference variables may denote the same Object instance. Variables of Value types exhibit value semantics: assignment means e.g. copying the value from one variable to another.

Data types can be defined in various ways: either by enumerating the elements of the type (literal) or by constructing a tuple from elements of given sorts (structure of fields).

Operations are Operators or Methods. Operators are functions that produce new objects or values, while Methods are applied to instances and can modify properties of the actual instances. Operations in general can be defined as virtual, redefined and finalized Operations.

In addition to Operations a DataType may also define local DataTypes. DataTypes can be defined as part of Packages and as part of types. DataTypes defined locally to a type can be defined as virtual DataTypes and thereby redefined/finalized in a subtype of the enclosing type.

**UML Mapping.** Object and Value data types are mapped to stereotyped Classes, i.e. the predefined UML DataType is not used. Variables of Object and Value types are therefore mapped to attributes with user-defined classes as attribute types. Fields of structured types are mapped to Attributes. Operators and Methods are mapped to a combination of Operations and Methods.

**Base Classes**
- Object type is a stereotyped Class ("Object");
- Value type is a stereotyped Class ("Value").

**Tags**
For both "Object" and "Value" the following Tags apply:
- Virtual - the object/value type is a virtual type;
- Redefined - the object/value type is a redefined type;
- Finalized - the object/value object/value type is a finalized type.

Operators and Methods have the same visibility options (public, protected, private) as Operations in UML, so there is a direct mapping.

In Figure 37 it has been decided to model Accounts by means of objects with fields and methods. The corresponding textual SDL partial definition (not including the bodies of the methods) is also included in the figure.

The SDL version of the data type is given in the textual form, contained in a text symbol, which will be part of a diagram. SDL also allows a class symbol as a partial specification of the data type, but in order to have it completely defined, the textual form is necessary.

Figure 38 gives an example of the definition of a simple value data type AccessCode in both UMLSDL and in SDL UML.

**12 Packages and Overall System Specification**
Both SDL and UML have the notion of package for the grouping of elements and of a topmost unit of specification. Figure 39 is a package diagram in SDL, using the signal types defined in the package ATMSignals and defining three block types.
**SDL: Packages and Systems**

A Package is a grouping of definitions, including type definitions but not instance definitions. A Package may also contain other Packages. A Package may use another Package, including all its definitions or just a subset of these. The Package Interface of a Package defines the elements of a Package that are visible outside the Package.

The complete SDL System Specification consists of a number of Packages and possibly a System. If no System is included, the System Specification is simply just a means to define a set of Packages.

A System may also use Packages, with the implication that the definitions in the Packages become visible as if they were defined in namespace enclosing the System namespace. Systems do not contain Packages. Decomposition of a System into parts is obtained by the general mechanism for structuring of Agents. The System is the special outermost Block Agent (see Figure 5), and as Agents may contain Agents (connected by Channels), a System may be decomposed into a number of Block Agents or a number of Process Agents, depending on the concurrency involved between the parts of the System.

**UML Mapping**. A Package is mapped to a Package constrained as specified below. The use of a Package is mapped to an import Dependency. The Package Interface is mapped to public visibility for the model elements to which the elements in the Package Interface are mapped.

A System Specification is mapped to a Model Package with a system-stereotyped class for the System Agent.

While a UML Model may have a number of Models, each representing a view on the modelled system, the corresponding SDL system Specification may only contain one view, represented by the System.

Note also that the UML notion of Subsystem is not used in the mapping. The reason for this is that parts of an SDL system are Block Agents. Blocks can be created dynamically, they have unique identifiers and they can have attributes and operations – and that is not possible with UML Subsystems.

**Constraints**

A System Specification can only have Packages and a System, i.e. no other Models.

The UML model corresponding to Figure 39 is given in Figure 41.

---

**13 Context Parameters**

An SDL type can be parameterised, so that it can be used in different contexts. Parameterised types are often defined in packages, as they are supposed to be usable in different systems. Context parameters are typically types, but they may also be variables and instance sets.
SDL: Context Parameters

A type in general can have formal context parameters. A formal context parameter represents a corresponding entity in the context (scope) of the type.

The types in Figure 43 can be parameterised by context parameters.

Depending on the kind of type, context parameters can be agent types, procedures, data types, signals and even variables. The simple rule is that entities that can be defined in the context of a given type also can be context parameter of the type.

When defining a new type based upon a parameterised type, the actual context parameters are types, procedures, data types, signals or variables in the scope where the new type is defined.

Context parameters are constrained, so that the specification of the parameterised type can be analysed independently of where it is used for defining new types.

UML Mapping. Context parameters are represented by TemplateParameters of the Class that represents the SDL type.

The provision of actual context parameters is represented by a binding. The binding as an explicit relationship does not exist in SDL. It is part of the definition of the type as a subtype of the parameterised type.

In SDL the context parameters are specified as part of the name of the type, see Figure 44.

In Figure 45 the corresponding ATM class in UML is a class with template parameters. The actual parameters are provided as part of the binding.

14 References
2 UML documentation on: www.omg.org/uml.
Summary of Z.109 SDL Combined with UML

This paper provides an introduction to the ITU Recommendation Z.109: SDL Combined with UML. The style is different from the one of Z.109, in the sense that SDL (and a conceptual UML of SDL) is the entry point. From this viewpoint it should be clear which subset of SDL is covered.

Z.109 provides a specialisation and restrictions of the following UML model elements, with an indication of the mapping to SDL:

- Packages represent packages in SDL;
- Models represent SDL specifications, each consisting of a set of packages and a system. Subsystems are not used; the structuring of systems into subsystems is in SDL covered by a special kind of objects (block agents) and thereby mapped to Composition in UML;
- Classes represent SDL types, with the following stereotypes representing the different kinds of entity types in SDL. The features of classes represent different SDL type properties, depending on the stereotype:
  - «system»;
  - «block»;
  - «process»;
  - «procedure»;
  - «interface»;
  - «object»;
  - «value»;
  - «state»;
  - «signal».
- State machines represent state machines of agents;
- A subset of Associations represents the corresponding concept in SDL. Two special kinds of association represent partially other SDL concepts:
  - composition as a partial representation of containment between agents;
  - association stereotyped with gate representing a gate with endpoint constraint;
- Generalization represents the corresponding specialisation in SDL;
- The following UML Dependencies are used to represent different dependencies in SDL:
  - «import»: a package using another package;
  - «create»: an agent being created by another agent;

Virtual types and procedures/operations are represented by the following Tags:

- virtual;
- redefined;
- finalized.