A Bottom-Up Abstract Data Type Editor

by Bernard Stepien
Telecommunications Software Engineering Research Group
University of Ottawa

Motivation

Formal description techniques enable to specify communications protocols in very concise and unambiguous ways. However, this implies learning yet another language with its syntax and semantic. This learning curve may actually prevent some uninitiated potential users to use such languages and revert to simpler but less powerful description methods. This has been widely the case for LOTOS (Language Of Temporal Ordering Specification) mostly due to its data description part that uses the Abstract Data Type language (ADT) Act One. For the same reason there has been a wide resistance to use the ADT language of the SDL tools that are however widely used for the procedural part of specifications.

The ADT languages Act One and its SDL equivalent
are however relatively simple and are based on some object oriented principles such as inheritance and polymorphism. This means that one may construct a new data type by re-using elements of an inherited type. Usually for complex specifications of more than 200 lines, users will get lost in these networks of inheritances. First of all, the user needs to remember all the details of the already defined data types and see how to integrate them in the new data type she is building. A very simple solution to this problem is to provide the user with a tool that presents what elements are available to build a new data type. This process varies at every level of a data type definition, inherited type selection, sorts definition, operations definition and finally equations definitions.

This method of ADT construction is using the bottom-up approach. This implies that one is not allowed to use something that has not been already defined. Other editors use the top-down approach and display error messages when an undefined element has been used. The bottom-up approach is with the help of this tool also usable in a step by step top-down oriented approach. The user need merely to build the skeletons of her intended data types and gradually complement them as more details are progressively defined. More research is currently undertaken to integrate the top-down with the bottom-up approaches as a result of the benefits of this tool.

Abstract Data Type language description

An abstract data type is composed of four components:

- the inherited data type reference
- the sorts definitions
- the operations definitions
- the equations definitions

an example:

```
type Bit is Boolean
    sorts Bit
    opns
        0, 1: -> Bit
        _eq_: Bit, Bit -> Bool
    eqns
        ofsort Bool
            forall X:Bit

            X eq X = true;
            0 eq 1 = false;
            1 eq 0 = false;
endtype
```

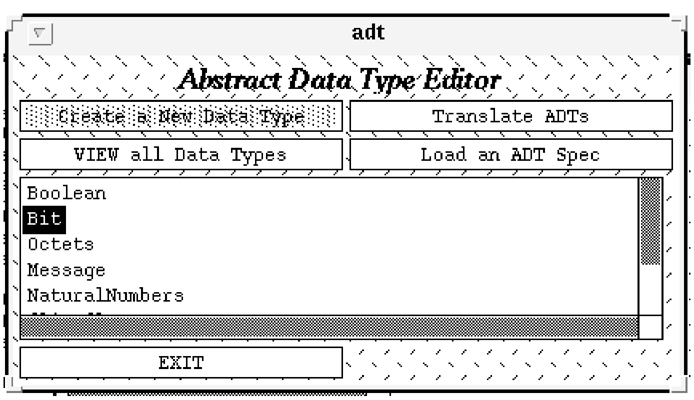
More complex data types can be constructed by renaming the elements of an existing data type or by actualizing a parametric data type.

The Abstract Data Type editor tool

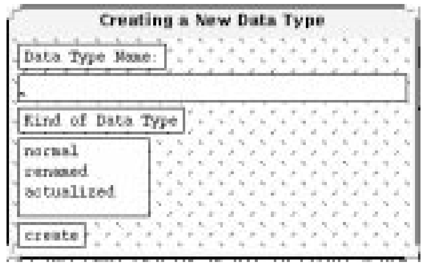
This Xwindow tool is composed of a number of interfaces:

- the main menu interface containing the list of already defined types
- the data type kind selection interface
- the data type definition interface
- the inherited data types selection interface
- the sorts definition interface
- the operation definition interface
- the equation definition interface
- the text translation viewing interface

The main menu interface



The data type kind selection interface

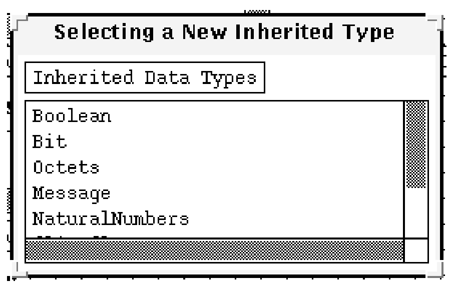


The data type definition interface

This is the master interface for a basic data type that is neither renamed or actualized. The user can invoke the appropriate interfaces pertaining to the four components of a data type by clicking on a text line to select an item to modify or clicking on the various buttons.

The inherited data types selection interface

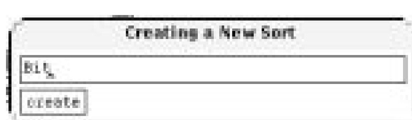
This interface can be invoked by clicking the “Add an Inherited Type” button.



The user can pick her desired inherited data type by clicking on one of the type names available in the list that has been generated by the software.

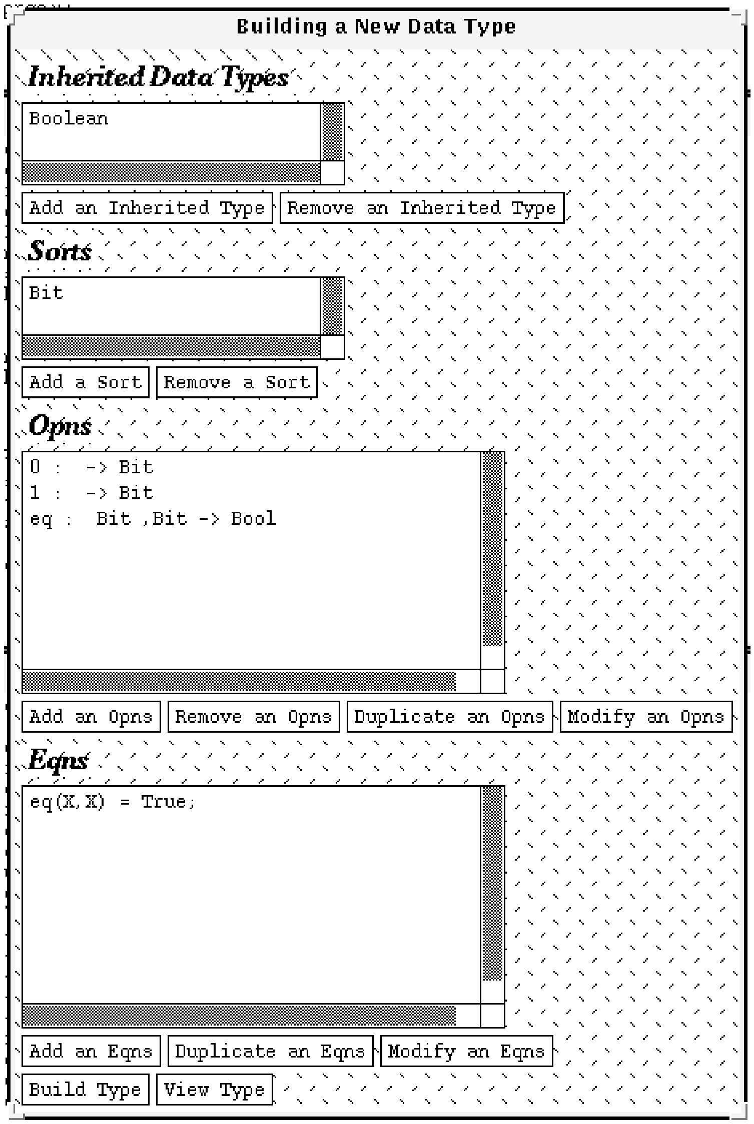
The sorts definition interface

The sorts definition interface is merely a text widget where the user enters a sort name.

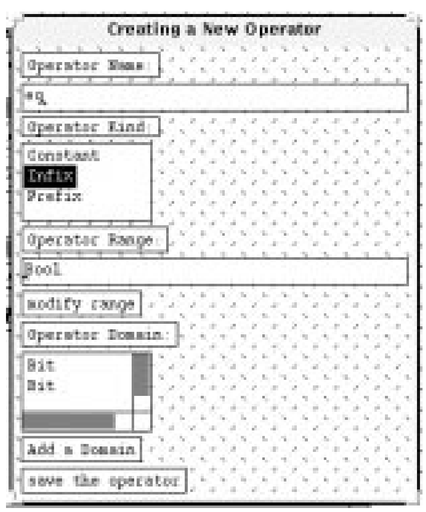


The operation definition interface

This interface can be invoked both for a new operator definition and an existing operator modification. The user enters a name in the text edit widget, clicks on one of the available operator kinds, clicks on the modify range button to obtain a list of available sorts to choose from. The same list of available sorts will appear when clicking the “add a Domain” button.



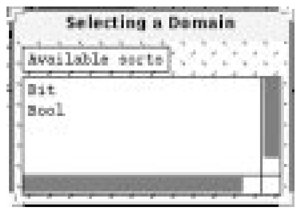
Building a New Data Type
Inherited Data Types
Boolean
Add an Inherited Type
Remove an Inherited Type
Sorts
Bit
Add a Sort
Remove a Sort
Opns
0 : -> Bit
1 : -> Bit
eq : Bit ,Bit -> Bool
Add an Opns
Remove an Opns
Duplicate an Opns
Modify an Opns
Eqns
eq(X,X) = True;
Add an Eqns
Duplicate an Eqns
Modify an Eqns
Build Type
View Type



Creating a New Operator
Operator Name:
Operator Kind:
Constant
Infix
Prefix
Operator Range:
Bool
modify range
Operator Domain:
Bit
Bit
Add a Domain
save the operator

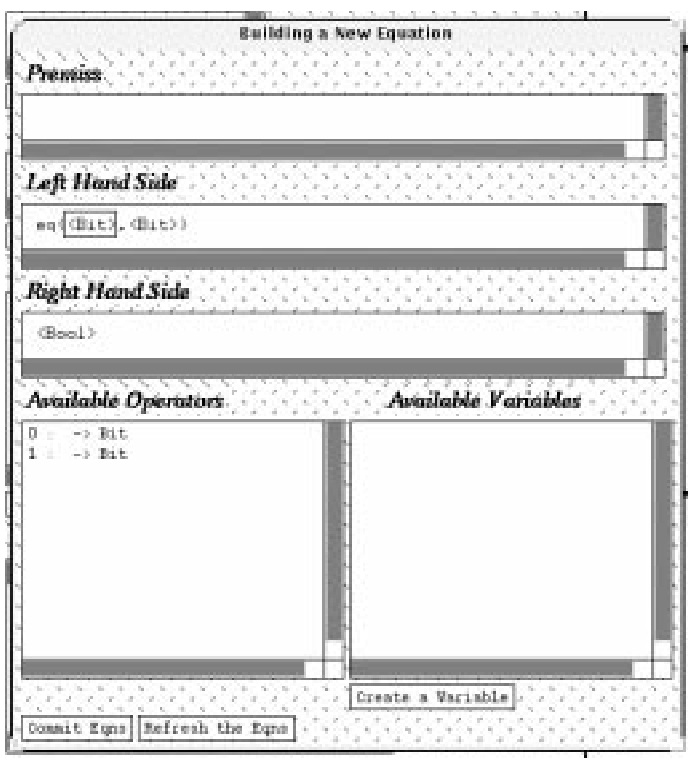
Range selection interface

Range selection interface

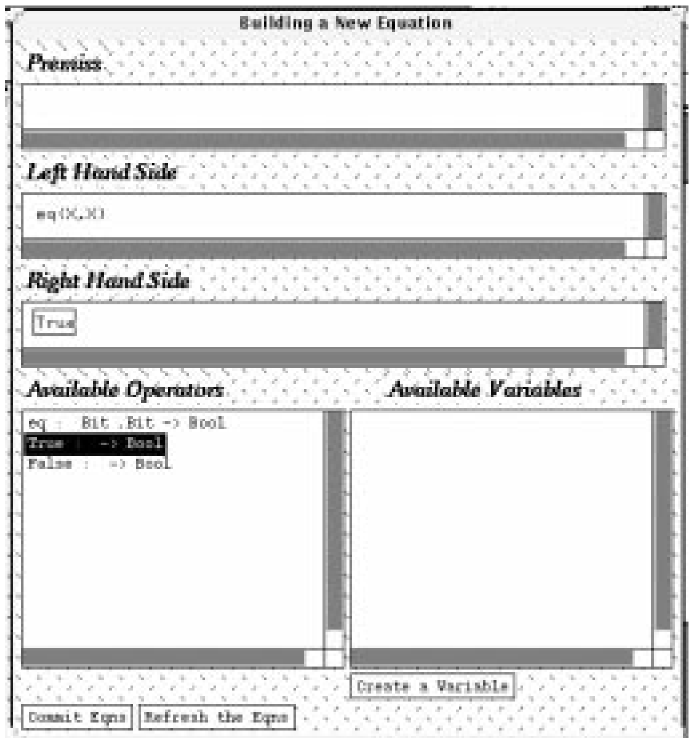


Equation definition interface

The most useful interface in this tool is the equation definition interface. Very complex equations can be built gradually by merely picking available operators that are displayed in list boxes.



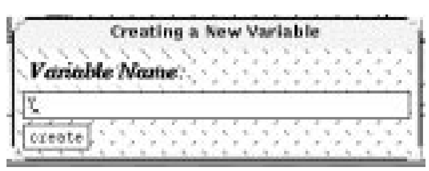
First the user must pick an operator from the operations list box of the Data type interface. Then she must click on the “Add an Eqns Button” to invoke the Building a New Equation interface. Initially, the tool displays the Equations interface filled with the name of the selected operator and the sorts of the domain and range of this operator that will appear as left and right hand side of the equation. The two list boxes “Available Operators” and “Available Variables” will display the available operators and variables or the domain or range sort that is framed. Clicking on one of the available sort names will automatically fill the framed domain element and move the frame to the next undefined element.



Complex equations that appear repeatedly with minor modifications can be duplicated in the Type definition interface and merely modified, thus saving even more efforts as for example for bits selectors in an octet:

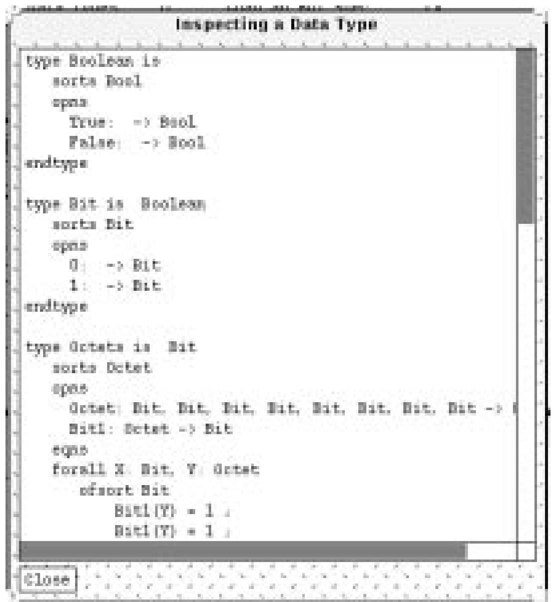
Bit1(Octet(b1,b2,b3,b4,b5,b7,b8) = b1;
Bit2(Octet(b1,b2,b3,b4,b5,b7,b8) = b2;
etc. for the remaining six selectors.

Equation variable definition interface



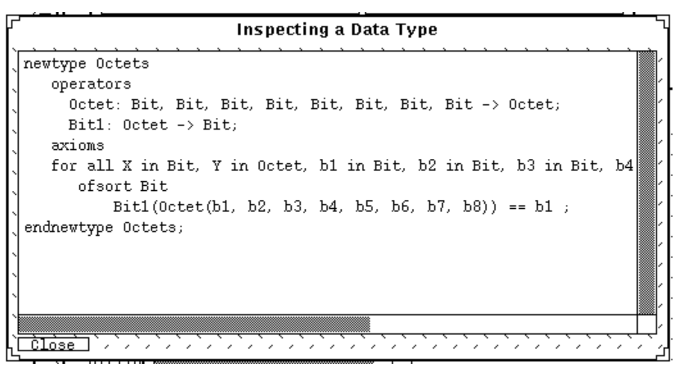
The text translation viewing interface

This type of interactive bottom-up tool can sometimes also confuse the user because the information becomes scattered in individual interfaces. A text translation interface can be invoked in various parts of the tool to display the full translation of the constructed abstract data type using either the LOTOS Act One of the SDL notation.



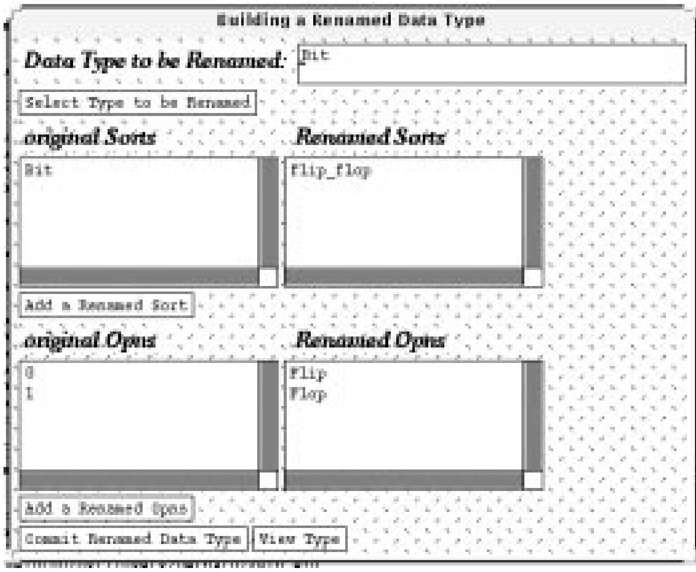
Three levels of translation can be generated:

- for a given type only
- for a given type and all its inherited types
- for all data types present in the module

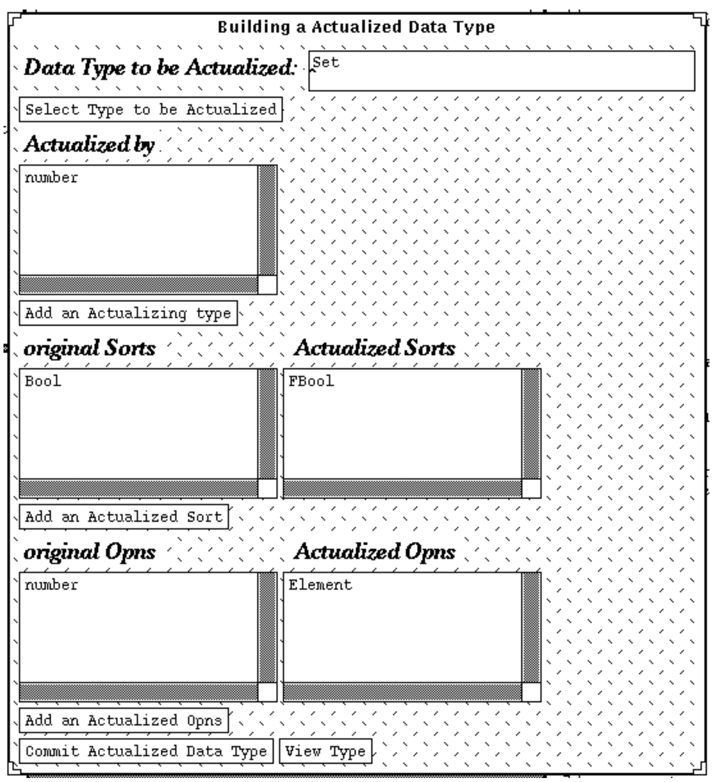


The text translation interface is also interactive. After browsing the text, the user can click on any line of that text and automatically invoke the full hierarchy of interfaces that leads to the interface required to modify that line.

Renamed data type interface



Actualized data type interface



Future developments

Extensions of the ADT language Act One are currently under study and have been subject to some experiments at GMD-FOKUS in Berlin. These extensions will be integrated in this tool. There is also a need to be able to manipulate architectural concepts in data type building.

Full extention to the SDL ADTs.

Contacts:

Bernard Stepien

183 Crestview Road
Ottawa, ON K1H 5G1
tel. (613) 733-3196
fax (613) 733-6783
E-mail: bernard@csi.uottawa.ca

Luigi Logrippo

University of Ottawa
Telecommunications Software Engineering Research group
Ottawa, ON K1M 6N5
tel. (613) 564-5450
fax (613) 564-9486
E-mail: luigi@csi.uottawa.ca