Use of UML to describe models of the ROS

(Last modified: 16-Feb-01) *(R.Cranfield, J.Vermeulen)*

Purpose

The purpose of this document is to present a proposal for the description of models to be used in performance and scaling studies of the ATLAS TDAQ ROS. The essence of this proposal is to use the diagramming techniques of the Unified Modelling Language (UML).

Motivation

The goal is to find a method for describing ROS models that is not committed to a specific modelling tool, but which can nonetheless be readily used with at least either of the main tools currently employed: Simdaq++ and Ptolemy. The description must be complete enough to include the features of the ROS that need to be modelled. It must also be accessible to all members of the ROS development team so that they can agree on the accuracy of the model. The whole point is to find a common "language" in which the members of the team can discuss and evolve the model.

Approach

The proposal is based on the use of UML (Unified Modelling Language) to describe the model. Note that this is subtly different from e.g. using UML to describe/model the ROS software itself. Not all elements of the software are relevant to the kind of model considered here, which is essentially for the purpose of studying the performance and scaling characteristics of the system. Nor is it likely that the UML description will be used to directly generate software for the implementation of the model since this software will in general contain many elements that, though necessary for running the model, are not essentially related to the model itself.

Ambiguity of the term "model"

Use of the term "model" can get confusing, particularly when talking about the UML which itself refers to a modelling process. In UML terminology the UML description of a system is referred to as a "model", which is why UML is called a "modelling" language. What we are concerned with in the current proposal is describing a discrete-event model (simulation) of the real-time behaviour of the ROS by means of the UML i.e. making a (UML) model of a (discrete-event) model. Normally we will use the term "model" to mean the latter (discrete-event model).

UML diagrams

A UML description appears mainly as a series of diagrams of different types, each showing a particular aspect (*view*) of the system being described. The diagrams themselves incorporate symbols that refer to elements of the underlying description (UML "model"). The UML is object-oriented and is typically used for the modelling of software written in e.g. C++ or Java; this is reflected in the fact that a primary UML diagram is the *class* diagram. Nevertheless the UML is intended to be applicable to a wide range of systems and provides several other types of diagram, some of which are particularly suited to the description of real-time systems. Indeed a full description of a system is likely to require the use of all of the different diagram types. Altogether the UML defines nine different types of diagram; below we show examples of five of these: **class** diagram, **sequence** diagram, **collaboration** diagram, **activity** diagram and **state** diagram.
In case this list seems intimidating it is worth noting that UML diagrams and symbols were intentionally designed to be relatively intuitive and easy to understand: creating the diagrams requires precise use of the appropriate elements, but reading them is hopefully fairly straightforward once the basic approach has been appreciated.

UML tools

The diagrams in the current document were produced using the JAVA-based, UML development tool **Together**. This tool has been reviewed favourably in the ATLAS offline environment and is available free of charge to academic users. It provides a reasonably comprehensive set of diagrams symbols and UML elements as described for instance in "The Unified Modeling Language User Guide"[] and …[]. Being Java based, it runs on several platforms and seems to be a good candidate for a general

diagramming tool for describing models of the TDAQ Dataflow – this despite the fact that it is very memory hungry and is a little glitchy on Linux.

Documentation

The current document was produced by simply inserting individual diagrams produced in Together into text in a WORD document. For future development work, however, the documentation will need to be readily updateable. Together has several document production options to facilitate this: the panels used in the tool itself can be output as “active” Web pages; the diagrams and element information can be written direct to a printer or PDF file in standard format; a standard templated report can be produced in the form of an RTF file with associated picture files; or the report template can be customised. Further experimentation is required to establish the best option in our case.

Diagram examples

There follow a series of example diagrams, illustrating the use of the different diagram types. These example diagrams show one part of the DAQ-1 DataFlow software, a so-called IOM task with name TRGTask6 which runs on the TRG [].

Class diagram

The class diagram shows the classes of the system as boxes, with various kinds of linkage between them. If we were directly describing a piece of object-oriented software the classes would simply be the e.g. C++ or Java classes defined in the code. However, in our case, we are trying to describe a model which is a model of both software and hardware; moreover, the software is currently not written in an object-oriented language (though it is based on a modular, high-level design). What we need to do is to provide an object-oriented analysis of the significant features of the ROS to identify the relevant elements of a class diagram for the system.

Example:

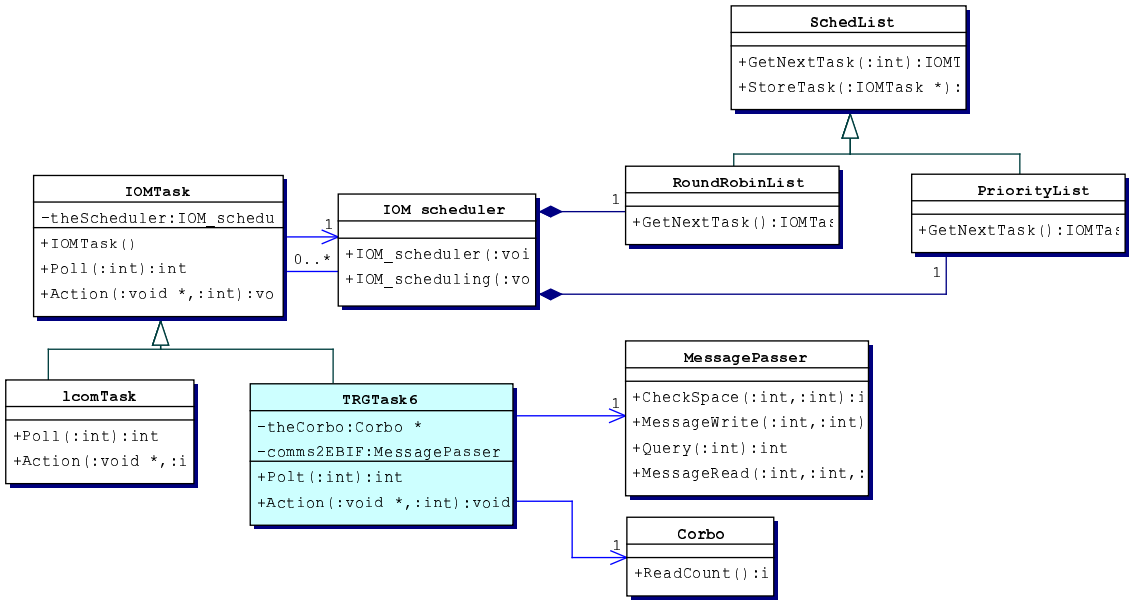

Figure 1. Class diagram for the TRG

Figure 1 shows an example of such an analysis of the main operation of one component of the current ROS Phase-1 software.
It illustrates how the TRGTask6 and lcomTask can be seen as special case classes (inheriting from) a general IOMTask class (the linkage with an open unfilled arrowhead indicates specialisation). The classes themselves are shown as boxes with a name and two other sections; the first section lists the attributes (cf. C++ data members) of the class, whilst the second lists the operations (cf. C++ member functions). Operations of IOMTasks are invoked by the IOM Scheduler (Poll and Activate, in particular, being relevant for modelling).
The TRGTask6 is associated (communicates) with a Corbo (class representing a piece of hardware) and a MessagePasser, which takes care of communication with the EBIF. Numbers on the linkages indicate the possible multiplicities of the associations (e.g. 1 Corbo per TRGTask6).
In general an IOMTask is associated with a scheduler (IOMsched) which contains (filled diamonds) several lists as shown.
(The highest LVL1 Id seen by a ROB is placed by that ROB in the memory space of TRGTask6, though this cannot be seen in the class diagram).

Activity diagram

The class diagram provides information on the context in which the object to be modelled is functioning. After having established it, a natural step to take is to draw one or more activity diagram(s) for the object to be modelled.
The activity diagram describes the logic flow, for example of an individual class operation. It is essentially a form of flowchart.

Example:

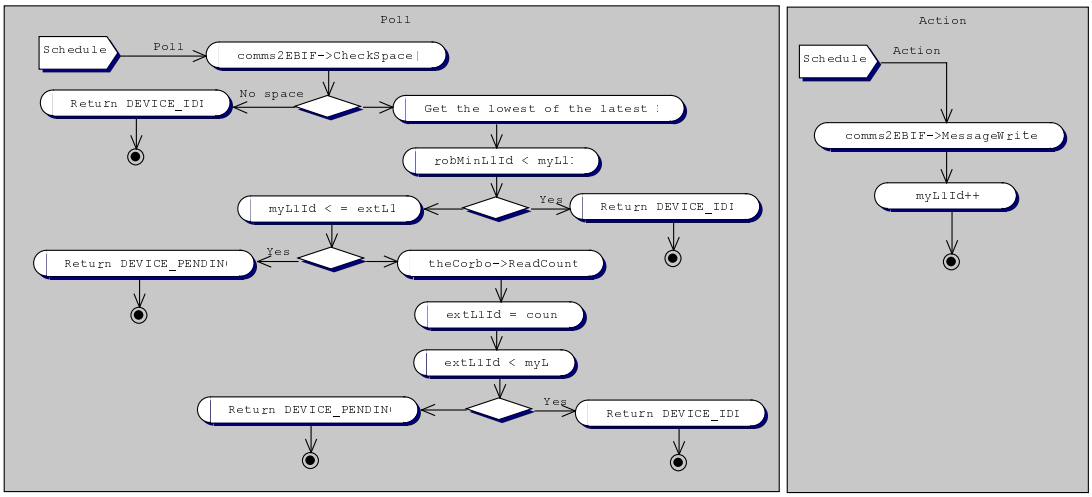

Figure 2. Activity diagram, showing flow of control for the Poll and Action operations of TRGTask6.

Collaboration diagram

The collaboration diagram shows objects (specific instantiations of classes) and the messages passing between them. It is clearly relevant to those parts of the system in which actual messages are passed between components, but the messages in the diagram can also refer to other interactions such as function calls (distinguished by different kinds of arrow symbol).
The objects in a collaboration diagram are assumed to be instantiated from classes in the class diagram, and the messages can be associated with the operations of the class of the destination objects. Additionally the messages have identifiers which indicate their position in the sequence of message interactions.

Example:

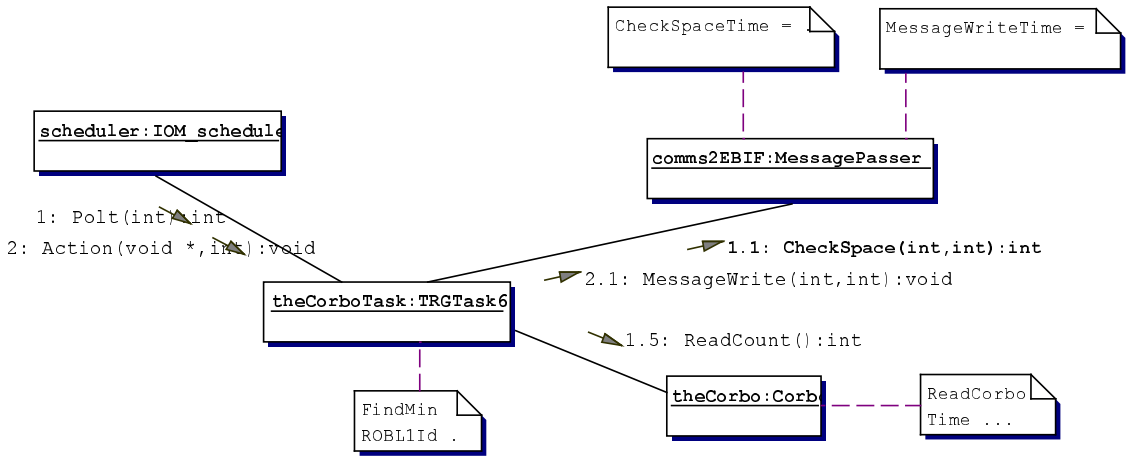

Figure 3. Collaboration diagram showing the interactions between the IOM scheduler, TRGTask6, the comms2EBIF object (class :MessagePassser) and the theCorbo object (class :Corbo). The notes specify where and how much time is spent.

The example shows an instantiation ("theCorboTask") of the TRGTask6 described in the earlier class diagram. It interacts with an IOMsched object ("scheduler") via a pair of messages (Poll and Action) which cause the invocation of the respectively named operations of the TRGTask6 class.
The boxes with the turned-down corners, attached by dashed lines, are just UML text notes, but the suggestion implied by the diagram is that we use these to indicate where operations that take time as far as the model is concerned.

Sequence diagram

The sequence diagram is closely related to the collaboration diagram in that one can be transformed into the other within a suitable UML tool. This is because the two diagrams share the same underlying entities. However the two diagrams emphasise different characteristics of the interactions between objects, and indeed show different details. The sequence diagram emphasises the sequence of interactions and has a column for each object, with a line descending vertically representing time; the periods during which the object is involved in an interaction are denoted by thickened sections of these time lines. Messages are represented by horizontal lines with arrowheads indicating the type of message. Timing information and conditions on the interactions are shown directly on the diagram.

Example:

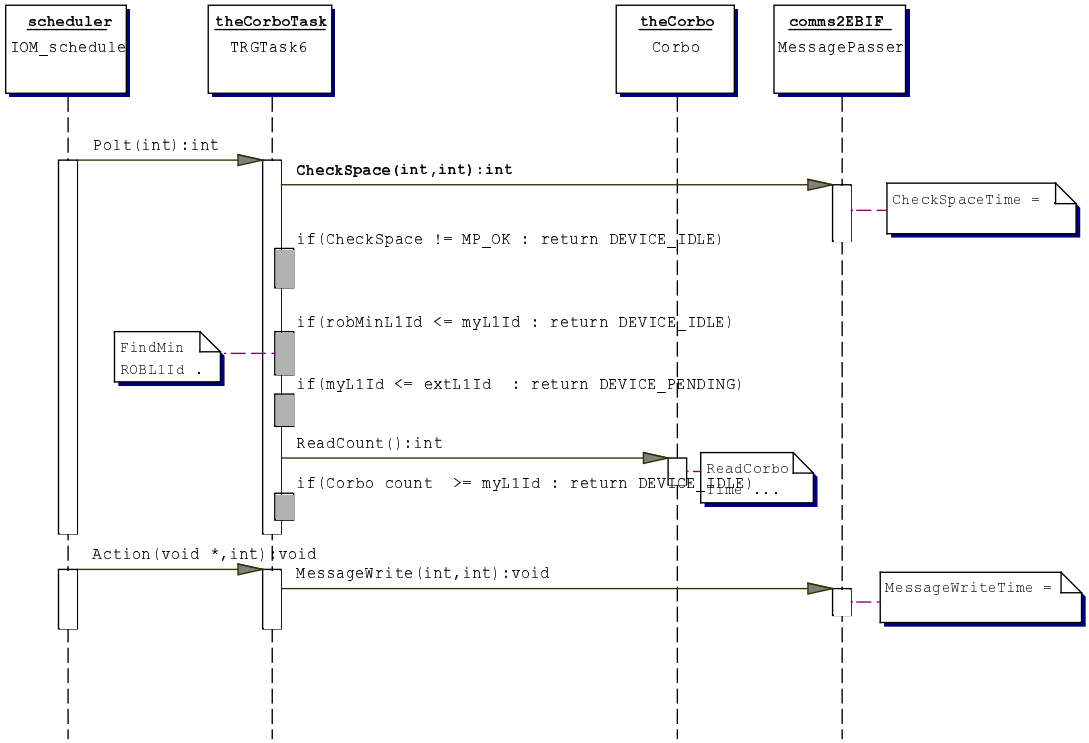

Figure 4. figure 3 shown as a sequence diagram, with notes showing where and how much time is spent

State diagram

The state transition diagram is another way of viewing the internal logic of operations. It shows the different states that an object can be in as round-cornered boxes, with arrowhead lines indicating the transitions between states. The events that cause these transitions are indicated by text annotations on the transition lines.
The state transition diagram is more relevant to the implementation of the model.

Example:

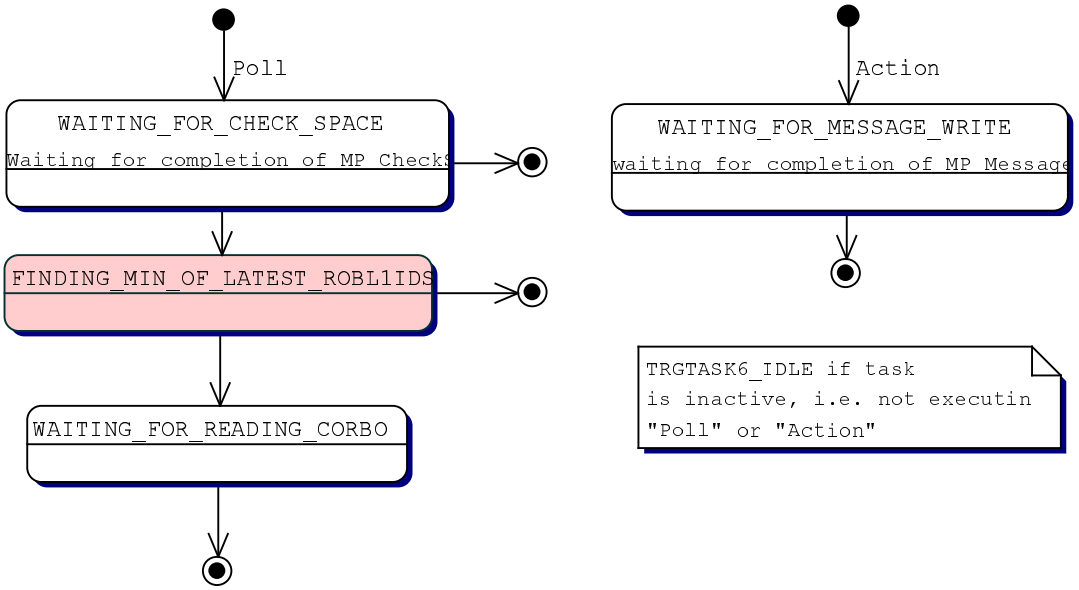

Figure 5. State transition diagram for the Poll and Activate operations for TRGTask6, to be used for implementation of the model.