Rule-Based Monitoring of Trigger-Rates at ZEUS

Ulf Behrens, Lars Hagge, Dirk Kahnert, Kars Ohrenberg
Deutsches Elektronen-Synchrotron, Notkestrasse 85, 22603 Hamburg, Germany

A tool for monitoring and surveilling the first-level trigger rates of the ZEUS experiment has been developed using a rule-based programming. The system has been designed with an object-oriented approach. Harel Diagrams have been used to model the behavior of trigger rates and the corresponding reactions on an alarm display. The approach is appropriate for fast implementation of well-designed and maintainable intelligent tools.

Key words: Diagnosis; Expert Systems; Monitoring Systems; Object-Oriented Methods; Programming Languages

1 Introduction

The global first level trigger of the ZEUS experiment combines 64 subtriggers into an overall trigger decision. Subtriggers flag eg energy thresholds, number of tracks, presence of muons, or topoligical event information.

Monitoring subtrigger rates is essential for efficient experiment operation. Currently, subtriggers are checked manually for nominal rates: for each slot, a two-dimensional histogram accumulates entries of current trigger rate versus current luminosity. Trigger rates are nominal if the entries remain within a given band (cf fig. 2).

Checking trigger rates manually is tedious and time consuming. It is a routine operation which is occupying far too much of the shift crew's attention. This papers demonstrates how object-oriented design and rule-based programming can be used for fast and efficient construction of an automatic monitoring and alarm system.

2 Object-Oriented Design of a Trigger Monitoring System

The trigger monitoring and alarm system has been designed following an object-oriented approach. First, classes were identified to encapsulate the different behaviors and objectives which contribute to trigger monitoring [?].

Fig. 1 shows the system architecture. The two major contributors are the subtrigger "slot" and the alarm "lamp". The slot focusses on classification and rating of subtrigger rates, the lamp encapsulates strategies how to react in specific situations. The other classes – beam, run control, global status lamp – are needed to interface the system with the environment.

Behavior is modeled using Harel diagrams. Harel diagrams are hierarchical state-transition diagrams. They offer intuitive visualization of behavior, are easy to understand and to communicate, and therefore an efficient way capture expert knowledge and incorporate it into programs.

Fig. 2 shows the Harel diagram for a slot. The rounded rectangles ("blobs") represent system states, arrows indicate state transitions. States may be nested, and arrows connecting to parent states are

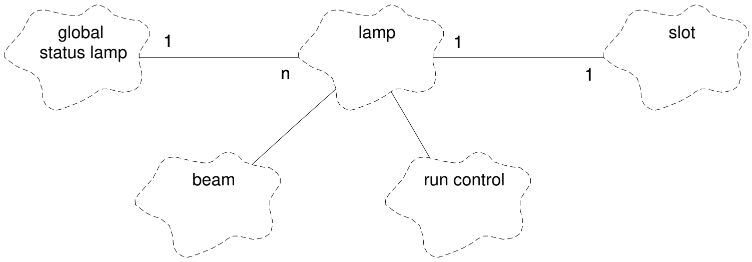

Fig. 1. Class diagram for the trigger monitoring system in Booch notation.

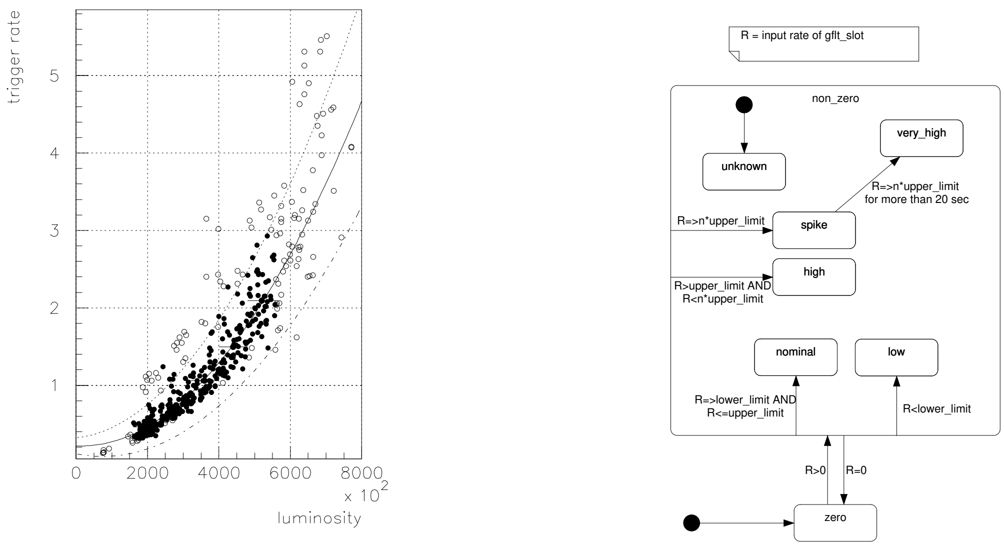

Fig. 2. Left: Trigger rate .vs. luminosity for a selected subtrigger. The trigger rate is nominal if the entry remains within the dashed limits. Right: Harel diagram for the slot class. The diagram defines the classification of trigger rates (cf text).

inherited by all substates. For example, if a slot is rated "very high" and the input rate suddenly drop below the lower limit, then the slot is immediately rated "low" because the transition connects to the "non-zero" state, which is parent to "very hight". Fig. 3 shows the Harel diagram for the trigger lamp. It defines how the monitoring system should react in case the trigger rate acquires anomal values.

An obvious benefit of Harel diagrams is that they can be incrementally refined. The object-oriented approach ensured that the behavior of the lamp is separated from the observation of the slot. The lamp, ie the alarming strategy, can thus be modified without touching the monitoring algorithms itself, which allows system evolution without introduction of unwanted side effects.

3 Rule-Based Implementation

The trigger monitoring system has been implemented with a rule-based inference engine. Harel diagrams map naturally to rules. The following rule implements for a slot the transition from parent state "non zero" to substate "high":

```
IF
        slot.?obj?.slot_status = non_zero
```

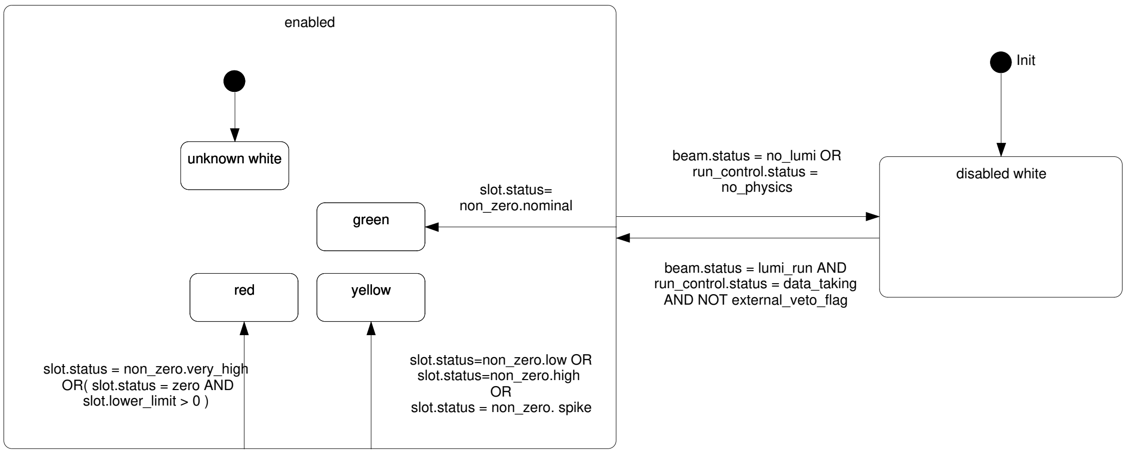

Fig. 3. Harel diagram for the lamp class defining the alarming strategy.

```
AND     slot.?obj?.slot_value <> high
AND     slot.?obj?.input_rate > slot.?obj?.upper_limit
AND     slot.?obj?.input_rate < slot_setup.?obj?.n_times_above *
                                         slot.?obj?.upper_limit
THEN
        slot.?obj?.slot_value := high;
        slot.?obj?.trigger_error_message
          :=concat(GetTimeStr(),": input_rate above upper limit");
!
```

The rule is in the syntax of a special inference engine. Variable names are in the form `class.object_id.attribute`. In the above case, `?obj?` is a wild card, which indicates that the rule should be tried for all known objects which belong to the class `slot`. It is a natural and very efficient way to account for the inherent parallelism into the monitoring system.

Using rule-based programming and Harel diagrams ensures that the source code corresponds directly to the documentation. The diagram could also be implemented in other languages like eg C++, but the big advantage of rule-based over conventional programming languages is that rules are not ordered. Conventional programs are sequential, and new rules would need to be inserted at the correct place. In rule-based environments, the inference engine checks all available rules for applicability before it decides which of them should be executed. Introducing new rules occurs eg when existing strategies should be refined, or when new situation should be incorporated into the system.

The system was implemented within two man-months using a commercial expert system shell. Data transport to/from the inference engine is implemented in C++. The system was introduced into the ZEUS environment towards the end of the 1996 running period.

4 Conclusion

Using rule-based programming has shown to be an easy and efficient way to map expertise and knowledge of human experts into small, helpful tools. Using an object-oriented approach and Harel diagrams for design ensures that the resulting system is well-partitioned and can relatively easy be maintained, updated, or evolved. Rule-based programming is a thing to look at, especially rule libraries for C++ are appearing on the market.