A Logic Programming Approach to Knowledge-Based Control Systems

Douglas Renaux* and Paul Dasiewicz †
dougren@vlsi.uwaterloo.ca dasiewic@vlsi.uwaterloo.ca

*Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada, N2L 3G1

†CPGEI / CEFET-PR
Centro Federal de Educação Tecnológica do Paraná
Rua Sete de Setembro 3165
Curitiba, PR, 80230

Abstract

RTX-Parlog (Real-Time Extended Parlog) is a concurrent logic programming language for real-time systems. It is specially suited for the implementation of Real-Time Knowledge-Based Control Systems. In this paper, the most relevant aspects of RTX-Parlog are described: time management, scheduling, and the tools for timing and scheduling analysis.

1 Introduction

The increasing complexity of control systems in areas such as Process Control, Computer Integrated Manufacturing (CIM), Flexible Manufacturing Systems (FMS), and Robotics, requires the use of state-of-the-art problem-solving techniques, such as the use of Real-Time Knowledge-Based Control Systems. In such applications, the control system executes a large number of processes concurrently. These processes have real-time constraints (deadlines). If deadlines are not met the system fails, possibly with catastrophic consequences. Conversely, knowledge-based systems make extensive use of search over complex knowledge bases and data bases, and consequently their response times have a large variance [PABS91]. This basic conflict between the two aspects of real-time knowledge-based control systems has been the subject of much research. Solutions to this conflict require the cooperation between the two fields (real-time systems and knowledge-based systems) to dynamically make trade-offs between [Sch91]: processing power, response time, data

---

*Supported by CEFET-PR/CNPq (Brazil) # 200492/89-6
†Supported by NSERC (Canada)
space, information inattention, and result degradation. There are many aspects to this problem: time constrained reasoning techniques, knowledge representation and partitioning, programming languages for real-time AI, computer architecture for real-time AI, and others. In this paper we focus on the programming language aspect. The use of the language presented here (RTX-Parlog) does not preclude the use of techniques developed for the other aspects above, rather, these techniques can be implemented in RTX-Parlog. These techniques include the use of knowledge areas (declarative procedure specification on how to accomplish goals or react to situations [IGR92]) and of knowledge partitioning [PABS91].

Programming languages used for non-real-time knowledge-based systems are usually one of functional (e.g. LISP), logic programming (e.g. PROLOG), production system (e.g. OPSS), or proprietary languages. These languages are preferred over conventional (imperative) languages because they are declarative and symbolic, facilitating knowledge representation and inference.

The problems with current tools and languages for knowledge-based systems when used in the real-time domain are [LG89] [LCS+88]:

- not fast enough
- minimal capabilities to reason over time
- difficult to embed
- difficult to integrate to conventional languages
- no real-time clock
- cannot handle asynchronous events
- not designed for continuous operation
- unpredictable response time

We extended the concurrent logic programming language PARLOG to provide support for real-time systems. RTX-Parlog (Real-Time Extended PARLOG) [RD93] is a declarative and symbolic language designed to solve the problems listed above. Its run-time system has a real-time priority and deadline based preemptive scheduler, processes have access to a real-time clock, the programming environment has a timing tool to analyze the execution time of processes and a simulator used for scheduling analysis and performance estimates; garbage collection allows the system to operate continuously and an external language interface (to C) is available.

2 RTX-Parlog

Concurrent Logic Programming languages describe a set of processes that execute concurrently, communicate via shared logic variables (communication channels), and have a dataflow synchronization mechanism (a process suspends if its input data is not available). Among the existing concurrent logic programming languages, PARLOG was selected since it allows an efficient implementation (mode directives reduce run-time overhead) and because of its sequential-and ('&') and
sequential-or (';') operators (that allow explicit sequentialization of processes). Like other concurrent logic programming languages, PARLOG has no notion of time, neither of process priority: ready processes are selected for execution using a LIFO scheduling.

In the next sections, the most relevant aspects of RTX-Parlog are described.

### 2.1 Time Management

RTX-Parlog has a built-in abstract data type `time`. Values of this type are absolute (date) or relative (duration) times specified in microseconds (the actual resolution of the current implementation is 10 ms). The operations on objects of type `time` are: plus (+), minus (-), less than (<), and larger than (>). The variable `Time` is a predefined variable of type `time` that represents the current time. An expression of the form `Time > T1`, where `T1` is an expression yielding an absolute time value, can either succeed (if the current time is later than `T1`) or cause the process to suspend until time `T1`. Expressions of this form are used to implement delays and to specify periodic processes.

A periodic process that polls a sensor every 50 ms and sends the value read to another process can be written in RTX-Parlog as:

```prolog
% type sensor_read(time, channel of sensor_value).
% mode sensor_read(?).% suspend until next period
sensor_read(PollTime, Channel) <-
  Time > PollTime, % end of the guard
  get_sensor(Value) & Channel = [Value|Channel1], % read the sensor and then...
  NextPollTime = PollTime + 50 ms, % send the value
  sensor_read(NextPollTime, Channel1). % calculate next poll time

% loop back
```

Figure 1: RTX-Parlog code for sensor polling process

### 2.2 Scheduling

The run-time system of RTX-Parlog includes a priority and deadline based preemptive scheduler. The scheduler has an array of ready queues: one queue for each priority level (Fig. 2). Ready processes are scheduled by priority first, and within a priority level the scheduling policy is one of: EDF (Earliest Deadline First), FIFO, or LIFO with time slicing. Process deadlines are considered only for processes at an EDF priority level. The programmer specifies the priorities and deadlines of each process, as well as the scheduling policies for each priority level, in a separate file, which is used as input to the RTX-Parlog compiler.
2.3 Scheduling Analysis

The programming environment of RTX-Parlog includes a timing tool and a simulator. The timing tool extracts from an RTX-Parlog program the execution times of the processes. The simulator inputs are:

- the timing information generated by the timing tool
- a description of how the input signals (service requests) are generated (request rates are specified as average values of Poisson distributions)
- a description of the processes activated by each input signal and of the inter process synchronizations
- the number of resources available (if applicable)

The execution of the RTX-Parlog program is then simulated, producing:

- a performance estimate of the system
- a timeline
- an analysis of the resource utilization and processor utilization, over time

The timeline is a list describing the sequence of events (inputs, outputs, and processes executed) and their occurrence times. The timeline is a useful tool for scheduling analysis, as the causes of missed deadlines can be identified and corrected.
3 Conclusion

RTX-Parlog is an extension of the concurrent logic language Parlog that provides support for real-time programming. Since RTX-Parlog is a logic programming language, it is specially suited for the implementation of real-time AI, such as, real-time knowledge-based control systems. RTX-Parlog design considered the limitations of non-real-time languages and tools, commonly used for real-time AI. It can be used in conjunction with many implementation techniques, such as, knowledge-base partitioning and knowledge areas. The most relevant aspects of RTX-Parlog were described: the abstract data type time, the real-time preemptive scheduling, and the timing and scheduling analysis tools.

Acknowledgements

The authors would like to thank the Parlog Group at the Imperial College (London), headed by Prof. Keith Clark, for kindly providing to us the source code of the JAM emulator and Parlog to JAM compiler, which we modified to implement RTX-Parlog. Many thanks to Damian Chu (Parlog Group) for his technical support.

References


