PRTccp: Priority-driven Real-Time Concurrent Constraint Programming

Tsuyoshi Okita

1 PRT-UML Methodology

Component-based development of real-time systems, such as Real-Time CORBA, is one of the most vivid research areas in object-oriented system. Our research [Oki01] [Oki02] focuses on the necessary extension of UML to support such a real-time component development. Although current UML [UML14] includes partly the description of periodic task, this is less efficient to describe real-time constraints that are cross-cut similar to aspect oriented programming [Kicz97] because of their end-to-end nature. Few efforts have been made for supporting this real-time component-based development other than UML organization themselves.

We are proposing PRT-UML methodology for supporting real-time and parallel systems, based on orthogonal analysis to object-oriented systems. For the real-time systems, we provide concurrency diagrams. Our aims are 1) to describe real-time constraints, and 2) to provide scheduling policy independent way, 3) to provide flexible simulation environment.

The figure 1 is a summary of a concurrency diagram. As real-time constraints are often crossing over objects, we introduce a virtual thread to describe such an end-to-end real-time constraints. We make distinction between WCET (Worst Case Execution Time) and real-time constraints. The left part of the figure shows the notation of arrival and constraints pattern. There are four arrival patterns: periodic, sporadic, aperiodic, and init vthread, while there are five constraint patterns: 1) classical deadline constraint, 2) firm quality imprecise constraint, 3) firm deadline imprecise constraint, 4) no constraint, and 5) temporal distance constraint. Vertical axis is a time axis and its length means duration.

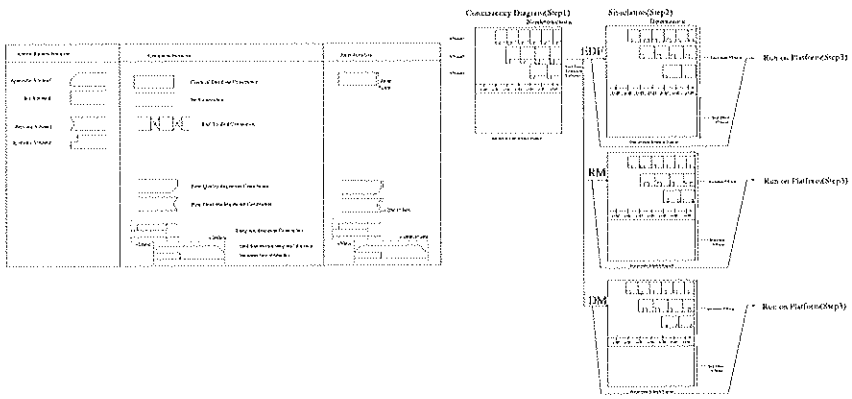

Figure 1. Concurrency Diagram Notation Summary (Left) and Overview (Right)

2 Real-Time Formal Language and Concurrent Constraint Programming

PRTccp is one of the precedence of ccp (concurrent constraint programming) [Sar87]. Tccp [Boer00] is a real-time extension of this ccp in order to describe a reactive system, where we claim that the Tccp approach is not enough to describe

priority-driven real-time systems. Tccp introduces the clock concept and non-determinisim to the concurrent constraint programming in order to handle a real-time system. Although tccp is intended for real-time system, to be precise, their intention is not a priority-driven system but a reactive system. In reactive systems, 1) there is an assumption that most of the calculation is finished within one clock, 2) ask / tell operation is finished within one clock, and 3) scheduling are always the best answer. However, priority-driven system is different in the following sense: 1) The calculation needs more than one clock, 2) ask / tell operation needs no negligible clocks, and 3) scheduling is not the best but just optimal and is decided by force by scheduling policy.

For the third one (scheduling policy), next figure 2 explains the difference. If the state S1 is defined as the state transition from the state S0 in the case of hardware, it transits from the state S0 to S1 at time X=0. However, in the case of software that has a scheduler, there is no guarantee that the targeted state is achieved. All the behaviors are controlled with the scheduler in the operating system. All the processes are queued in the scheduler to ask for its permission.

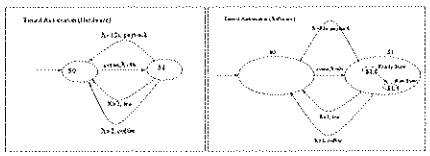

Figure 2. Timed automaton in hardware and (priority-driven) software

3 PRTccp

PRTccp bases on tccp [Boer00] and introduces a problem solver and priority. The former is a similar approach by Gupta enlarging ccp in their 'ask' operation to consume constraints. Original ccp has no concern about solving its constraint and the computation agents throw a constraint to the global store and wait until it is solved. This problem solver signifies a scheduler in a priority-driven system. The latter is the approach taken practically by real-time schedulability analysis [Liu00]. Although real-time constraints are visible to human beings, priority-driven system uses priority instead of real-time constraints and this conversion is vital to calculate using schedulability analysis.

A problem solver can access to the global store asynchronously and if agent store constraints, in the next clock, a problem solver knows that the constraint is increased. We introduce two types of constraint: Tc (real-time constraints) and Sc (synchronization constraints), where we assume that problem solving is assumed to consume time. Non-determinism of problem solving is shown by local choice. Sets of constraints have priority, where the entailment of constraint subjects to this. A virtual thread tell to the global store that it has to resolve t_{exec} unit of computation and it has to be done within $t_{end-to-end}$. When a problem solver entails this constraint, t_{exec} is decreased. Each Tc has a global timer that is set at release time and it timeouts when it reaches $t_{end-to-end}$. The main body of Sc is a shared variable between virtual threads. When the condition of Sc is satisfied, it is possible to reduce a constraint of Tc.

Definition 1 (PRTccp) *Assuming a given constraint system C the syntax of agents is given by the following grammar,*
$A ::= stop \mid error \mid \Sigma_{i=1}^{n} ask(c_i) \rightarrow A_i \mid tell(c) \mid now\ c\ then\ A\ else\ B \mid A \parallel B \mid \exists_X A \mid p(t) \mid A; B \mid A \propto priority \mid ProblemSolver.$ $\quad c ::= Sc \mid Tc.$
where the c, c_i are supposed to be finite constraints in C.

The operational model of PRTccp is described by a transition system where each transition step takes one time unit, where there are three kinds of transition: 1) congruence, 2) unprioritized timed transition, and 3) prioritized timed transition.

Definition 2 (Prioritized Timed Transition) $priority(T_{c1}) < priority(T_{c2}), (T_{c1} \oplus T_{c2}) \propto priority(T_{c1}), \xrightarrow{p} (T_{c1} \oplus T_{c2}) \propto priority(T_{c2})$.

The next example shows how a rate-monotonic scheduler is described using PRTccp.

Example 1 (Rate-Monotonic Scheduler) *In a rate-monotonic scheduler, task of short period has a higher priority. Each vthread has a static priority. When all the constraints are solved, RMSystem stops.*

$$
\begin{aligned}
&\mathrm{RM}System = \mathrm{VThread_1} \| \mathrm{VThread_2} ... \| \mathrm{VThread_n} \| \mathrm{RMScheduler};\\
&\mathrm{VT}hread_i(TC_i) = (\mathrm{delay(Tc_i)}; \mathrm{tell(Tc_i)}; \mathrm{ask(Tc_i)} \rightarrow \mathrm{stop}) \propto \mathrm{Priority_i};\\
&\\
&\mathrm{RM}Scheduler(Priority_{scheduler}, Tc_n \oplus Tc_l ... Tc_m) =\\
&\quad (Priority_{scheduler} < max(Priority_n, Priority_l, ..., Priority m) \xrightarrow{p,0}\\
&\quad RMScheduler(max(Priority_n, Priority_l, ..., Priority m), Tc_n \oplus Tc_l ... Tc_m);\\
&\quad [PRIORITIZED\ TRANSITION]\\
&\quad +\\
&\quad (Priority_{scheduler} == max(Priority_n, Priority_l, ..., Priority m) \xrightarrow{t,1}\\
&\quad RMScheduler(Priority_{scheduler}, Tc_n \oplus Tc_l ... Tc_m); [TIMED\ TRANSITION]\\
&\\
&\mathrm{Tc_i}(Priority_i, phase_i, endToEnd_i, exec_i, elapse_i) =\\
&\quad (Priority_{scheduler} == Priority_i, exec_i <> 0, elapse_i <> endToEnd_i) \rightarrow\\
&\quad Tc_i(Priority_i, phase_i, endToEnd_i, exec_i - 1, elapse_i + 1); [DISPATCH]\\
&\quad +\\
&\quad (Priority_{scheduler} <> Priority_i, exec_i <> 0, elapse_i <> 0) \rightarrow\\
&\quad Tc_i(Priority_i, phase_i, endToEnd_i, exec_i, elapse_i + 1); [NO\ DISPATCH]\\
&\quad +\\
&\quad (elapse_i == endToEnd_i) \rightarrow error; [TIMEOUT]\\
&\quad +\\
&\quad (exec_i == 0) \rightarrow stop; [JOB\ COMPLETE]
\end{aligned}
$$

4 Conclusions

This paper presents real-time formal language PRTccp, which complements priority-driven concerns in Tccp. First, we showed the necessity of formal language for priority-driven system compared to reactive real-time system. Secondly, we showed the grammar of PRTccp. Thirdly, we showed a small example of scheduler.

References

[Oki01] "Component-based Development of Embedded Software," Okita, T., Master Thesis, Vrije Universiteit Brussels, 2001.

[Oki02] "UML Extension for Real-Time Parallel Computation : Parallel and Concurrency Diagram," Okita, T., 27th IEEE/NASA Software Engineering Workshop, 2002 (submitted).

[Kicz97] "Aspect-Oriented Programming," Kiczales, G., Lamping, J., et.al., ECOOP, 1997.

[Liu00] "Real-Time Systems," Liu, J.S., Prentice Hall, 2000.

[Boer00] "A Timed Concurrent Constraint Language," Boer, F.S., Gabbrielli, M, Meo, M.C., Information and Computation, 2000.

[Sar87] "Concurrent constraint programming," Saraswat, V.A., ACM Doctoral Dissertation Awards, MIT Press, 1993.

[UML14] "OMG Unified Modeling Language Specification", Object Management Group, Version 1.4 beta R1, November, 2000.