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.

![Concurrency Diagram Notation Summary (Left) and Overview (Right)](image)

2 Real-Time Formal Language and Concurrent Constraint Programming

PRTccp is one of the precodence of ccep (concurrent constraint programming) [Sar87]. Tccep [Boer06] is a real-time extension of this ccep in order to describe a reactive system, where we claim that the Tccep approach is not enough to describe
priority-driven real-time systems. Tcep introduces the clock concept and nondeterminism to the concurrent constraint programming in order to handle a real-time system. Although tcep is intended for real-time systems, 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/cell operation is finished within one clock, and 3) scheduling is always the best answer. However, priority-driven system is different in the following sense: 1) The calculation needs more than one clock, 2) ask/cell 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 PRTcep

PRTcp is based on tcep [Boc00] and introduces a problem solver and priority. The former is a similar approach by Gupta enlarging cfp in their 'ask' operation to consume constraints. Original cfp 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 systems use 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: Te (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 texec unit of computation and it has to be done within tend-to-end. When a problem solver entails this constraint, texec is decreased. Each Tce has a global timer that is set at release time and it timeouts when it reaches tend-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 Tce.

Definition 1 (PRTcp) Assuming a given constraint system C the syntax of agents is given by the following grammar:

\[
A ::= \text{stop} | \text{error} | \sum_{c \in C} \text{ask}(c) \rightarrow A_i | \text{tell}(c) | \text{now c then A else B} | A || B | \exists X A | p(t) | A; B | A \propto \text{priority} \]

Problem Solver:

\[
c ::= Sc | Tce.
\]

where the c,ci are supposed to be finite constraints in C.

The operational model of PRTcep 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(Tc1) < priority(Tc2), (Tc1 ⊕ Tc2) ∝ priority(Tc1), \( Tc1 ⊕ Tc2 \) ∝ priority(Tc2).

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

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, RMSys stops.
RMSystem = VThread1 ||VThread2 || ... ||VThreadn || RScheduler;
VThreadi(TCi) = (delay(TCi); tell(TCi); ask(TCi) → stop) @ Priorityi;

RM Scheduler(PriorityScheduler, TC0 ⊕ TC1 ⊕ ... ⊕ TCm) =

(PriorityScheduler < max(Priority1, Priority2, ..., Prioritym) ⊕ 0)
RM Scheduler(max(Priority1, Priority2, ..., Prioritym), TC0 ⊕ TC1 ⊕ ... ⊕ TCm);
[PRIORITIZED TRANSITION]
+
(PriorityScheduler = max(Priority1, Priority2, ..., Prioritym) ⊕ 1)
RM Scheduler(PriorityScheduler, TC0 ⊕ TC1 ⊕ ... ⊕ TCm);
[TIMED TRANSITION]

TCi(Priority, phasei, endToEndi, execi, elapsedi) =

(PriorityScheduler = Priority, execi <> 0, elapsedi <> endToEndi) →
TCi(Priority, phasei, endToEndi, execi - 1, elapsedi + 1); [DISPATCH]
+
(PriorityScheduler <> Priority, execi <> 0, elapsedi <> 0) →
TCi(Priority, phasei, endToEndi, execi, elapsedi + 1); [NO DISPATCH]
+
(elapsedi = endToEndi) → error; [TIMEOUT]
+
(execi = 0) → stop; [JOB COMPLETES]

4 Conclusions

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

References


