Final Report

Title:
Enhanced Specification and Verification for Timed Planning

Principal Investigator:
Dr. Jin Song Dong, Co-PI: Dr. Jun Sun,
Researcher: Xian Zhang
School of Computing, National University of Singapore

Contract Number: FA4869-08-1-4006

AFOSR/AOARD Reference Number: AOARD-08-4006

AFOSR/AOARD Program Manager: Dr. Hiroshi Motoda

Period of Performance: 12 Dec 2007 - 12 Dec 2008

Submission Date: 28 Feb 2009

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
10 MAR 2009	**FInal**	**12-12-2007 to 12-12-2008**

4. TITLE AND SUBTITLE
Enhanced Specification and Verification for Timed Planning

5a. CONTRACT NUMBER
FA48690814006

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Jim Song Dong

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National University of Singapore,3 Science Drive 2,Singapore 117543,Singapore,SP,117543

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AOARD, UNIT 45002, APO, AP, 96337-5002

10. SPONSOR/MONITOR'S ACRONYM(S)
AOARD

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AOARD-084006

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In this project, the PI introduced a specification language named Timed Planning, which is an extension of Timed CSP with the capability of stating more complicated timing behaviors for processes and events. they also developed a reasoning mechanism for Timed Planning based on Constraint Logic Programming. They model the Pearl Harbor Attack plan to demonstrate the capability of their approach for modeling time based military plans with critical timing constraints. Their approach is capable to handle the extended job-shop scheduling problems. In their work, the job shop scheduling problems with extensions can be naturally modeled as Timed Planning processes, whose complete executions correspond to feasible schedules. By using CLP based reasoning mechanism, the optimal scheduler which is an execution with the minimum execution time, can be found.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**33**	

Objectives

Based on Timed-CSP, we aim to develop formal modeling techniques and tool for timed planning and apply them to multi-contexts domain.

Status of effort

We have extended Timed CSP and developed a reasoning tool that has applied to millitary planning, extended job scheduling and timed reminding systems.

Abstract

In this project, we introduce a specification language named Timed Planning, which is an extension of Timed CSP with the capability of stating more complicated timing behaviors for processes and events. We also develop a reasoning mechanism for Timed Planning based on Constraint Logic Programming. We model the Pearl Harbor Attack plan to demonstrate the capability of our approach for modeling time based military plans with critical timing constraints. Our approach is capable to handle the extended job-shop scheduling problems. In our work, the job shop scheduling problems with extensions can be naturally modeled as Timed Planning processes, whose complete executions correspond to feasible schedules. By using CLP based reasoning mechanism, the optimal scheduler which is an execution with the minimum execution time, can be found.

Personnel Supported

Dr. Sun Jun and Ms. Zhang Xian

Publications

J. S. Dong, J. Sun and X. Zhang, em Reasoning Timed Planning based on Constraint Solving. Formal Methods in System Design, 2009 (submitted)

Interactions

Participation/presentations at meetings, conferences, etc

presentation: at Formal Methods Workshop with Sir Tony Hoare, Nov 2008

Potential technology application

Potential applications include millitary planning, extended job scheduling and timed reminding systems.

New

List discoveries, inventions, or patent disclosures

Developed the verification tool for Timed Planning which is an extension to Timed CSP.

Completed the DD Form 882

See attached.

Honors/Awards

In 2008, PI (J S Dong) was invited as a Visiting Professor at National Institute of Informatics (NII), Japan (with an aunual fund that supports regular visit to NII).

Archival Documentation

See the following detalied technical report.

Software and/or Hardware

The updated tool HORAE will be made public: `http://comp.nus.edu.sg/~zhangxi5/horae`

Technical Report

Dr. Jin Song Dong (PI), Dr. Jun Sun (Co-PI), Xian Zhang (Researcher)

School of Computing,
National University of Singapore
{dongjs,sunj,zhangxi5}@comp.nus.edu.sg

1 Introduction

Timed planning is to schedule a set of given timed tasks to fulfill certain desired properties. It has important implications in a variety of domains, e.g., real-time operating system, military planning, etc. In a broad view, Timed Planning is a generalization of the well-understood schedulability problem of a given set of timed tasks [3].

In order to handle the generalized Timed Planning problem, in this work we model a timed task as a timed process specified using the well-established timed specification language Timed CSP [5]. A task may have complicated timed behavior patterns. Timed CSP is chosen over other timed formalisms (like Timed Automata [2], Task Automata [8], etc.) is because of its compositional and event-based nature. In order to state more timing behaviors of a timed task, such as deadline and period of a task and time-related constrains among events, we extend Timed CSP with capabilities to state timed-related system requirements in a modular manner. Namely, each process, which is a task or a sub-task, may be associated with a set of localized timing requirements. The Timed Planning problem is then reduced to a feasibility test of the tasks. Based on the notion of constraint solving [12, 13], we have developed a fully automated reasoning engine which clarifies whether it is possible that a given set of tasks are scheduled in a way to fulfill the timed requirements and generate a feasible scheduling (if possible). Besides, feasibility checking, as well as various property verification can be applied to Timed Planning specifications. Feasibility checking helps users to debug the conflicts of the timing constraints specified in the systems.

We study a real military attack, the Pearl Harbor Attack, which is a carefully planned military strike conducted by the Japanese navy. The whole mission consists of several sub-missions, each has its own duty and timing constraints. We model the main attack plan conducted on December 7, 1941. It is only a sub plan of the whole pearl harbor attack plan which also consists of preparation plans and follow-up plans after the main attack.

In this project, we apply Timed Planning to the classic scheduling domain, job-shop scheduling problem (both deterministic and preemptive). This problem cannot be modeled and solved using pure Timed CSP specifications. We also apply our approach to handle the extended job-shop scheduling problems, where jobs can have more complex relations, such as a composition of operational behaviors with communications, and jobs with deadlines and relative timing

constraints, which no other current work are able to support. It is the main feature of our work differs from [1] which proposed a mechanism for modeling the job-shop scheduling problems in Timed Automata and hence finding the optimal solutions. The job-shop scheduling problem can be very naturally modeled in Timed Planning processes, whose executions correspond to feasible schedules. In this work, the job-shop scheduling problem can be reduced to a problem of finding a complete execution (an execution that terminates) with the minimum execution time.

Another application is Timed Reminding System, which is a reminding system design to generate real time scheduler for coordinating multiple reminders and remedying possible conflicts. One typical application is to design a reminding system for the dementias to help them doing daily activities such as take medication at the right time, remind them prepare lunch, turn off stove after cooking and etc. Our timed reminding system consists of multiple reminders. Reminders can be triggered at a specific time or by some events. Reminders can interrupt with each other due to the unexpected activities of the participant. The reminder which has been interrupted also can resume to the previous one if the interrupted one is ended before the expiry time.

The underlying reasoning mechanism is Constraint Logic Programming (CLP) [13], which has been successfully applied to model programs and transition systems for the purpose of verification [10, 15], showing that their approach outperforms the well-known state-of-art systems. In our previous work [6], we constructed a reasoning tool using CLP as an underlying reasoning mechanism for Timed CSP . In this work, we extend the reasoning mechanism to support the Timed Planning specification. Our approach starts with a systematic translation of the semantics of Timed Planning into CLP. The operational semantics is encoded to CLP, where a set of global and local variables need to be captured during the execution, then a set of safety and liveness properties be verified. We implement a prototype reasoning engine based on one of the established CLP solvers, CLP($\mathcal{R}$) [14]. CLP($\mathcal{R}$) is chosen for its support of real numbers and continuous time variables. A number of theories, libraries and shortcuts have been developed for easy querying and proving.

The remainders of the report are organized as follows. Section 2 briefly review the background of the work, i.e., the Timed CSP specification language and the Constraint Logic Programming paradigm. Section 3 discusses the syntax and semantics of Timed Planning, while Section 4 introduces the reasoning method for Timed Planning. Section 5 shows how we model the Pearl Harbor Attack plans in Timed planning specification. Section 6 presents how to use Timed Planning to solve job-shop scheduling problems. Section 8 concludes this report.

2 Overview

Timed CSP Language Hoare's CSP [11] is an event based notation primarily aimed at describing the sequencing of behavior within a process and the synchronization of behavior (or *communication*) between processes. Timed CSP extends

CSP by introducing a capability to quantify temporal aspects of sequencing and synchronization. Inherited from CSP, Timed CSP adopts a symmetric view of process and environment. Events represent a cooperative synchronization between process and environment. Both process and environment may control the behavior of the other by *enabling* or *refusing* certain events and sequences of events.

Definition 1 (Timed CSP). *A Timed CSP process is defined by the following syntax,*

$$\begin{aligned} P ::= \ & \textsc{Stop} \mid \textsc{Skip} \mid \textsc{Run} \mid e \xrightarrow{t} P \mid e : E \to P(e) \mid e@t \to P(t) \\ \mid \ & P_1 \,\Box\, P_2 \mid P_1 \sqcap P_2 \mid P_1 \ {}_X||_Y \ P_2 \mid P_1 \,|[\, X \,]|\, P_2 \mid P_1 \,|||\, P_2 \\ \mid \ & P_1;\ P_2 \mid P_1 \,\triangledown\, P_2 \mid P1 \overset{d}{\triangleright} P2 \mid \textsc{Wait}\ d \mid P_1 \,\triangledown\{d\}\, P_2 \\ \mid \ & \mu\, X \bullet P(X) \end{aligned}$$

$\textsc{Run}_\Sigma$ is a process always willing to engage any event in Σ. Stop denotes a process that deadlocks and does nothing. A process that terminates is written as Skip. A process which may participate in event e then act according to process description P is written as $e@t \to P(t)$. The (optional) timing parameter t records the time, relative to the start of the process, at which the event e occurs and allows the subsequent behavior P to depend on its value. The process $e \xrightarrow{t} P$ delays process P by t time units after engaging event e. The external choice operator, written as $P \,\Box\, Q$, allows a process of choice of behavior according to what events are requested by its environment. Internal choice represents variation in behavior determined by the internal state of the process. The parallel composition of processes P_1 and P_2, synchronized on common events of their alphabets X, Y (or a common set of events A) is written as $P_1 \ {}_X||_Y \ P_2$ (or $P_1 \,|[\, A \,]|\, P_2$). The sequential composition of P_1 and P_2, written as $P_1;\ P_2$, acts as P_1 until P_1 terminates by communicating a distinguished event ✓ and then proceeds to act as P_2. The interrupt process $P_1 \,\triangledown\, P_2$ behaves as P_1 until the first occurrence of event in P_2, then the control passes to P_2. The timed interrupt process $P_1 \,\triangledown\{d\}\, P_2$ behaves similarly except P_1 is interrupted as soon as d time units have elapsed. A process which allows no communications for period d time units then terminates is written as $\textsc{Wait}\ d$. The timeout construct written as $P_1 \,\triangleright\{d\}\, P_2$ passes control to an exception handler P_2 if no event has occurred in the primary process P_1 by some deadline d. Recursion is used to give finite representation of non-terminating processes. The process expression $\mu\, X \bullet P(X)$ describes processes which repeatedly act as $P(X)$.

The detailed illustration of each process can be found in [18]. The semantics of a Timed CSP process is precisely defined either by identifying how the process may evolve through time or by engaging in events (i.e., the operational semantics defined in [19]) or by stating the set of observations, e.g., traces, failures and timed failures (i.e., the denotational semantics as defined in [4]). In this work, Timed CSP is used to specify interactive timed tasks.

CLP Preliminaries Constraint Logic Programming (CLP [13]) began as a natural merger of two declarative paradigms: constraint solving and logic pro-

gramming. This combination helps make CLP programs both expressive and flexible, and in some cases, more efficient than other kinds of programs. The CLP scheme defines a class of languages based upon the paradigm of rule-based constraint programming, where CLP($\mathcal{R}$) is an instance of this class. We present some preliminary definitions about CLP.

Definition 2 (Atom, Rule and Goal). *An atom is of the form $p(\tilde{t})$, where p is a user defined predicate symbol and $\tilde{t}$ is a sequence of terms '$t_1, t_2.., t_n$'. A rule is of the form $A :- \tilde{B}, \Psi$ where the atom A is the head of the rule, and the sequence of atoms $\tilde{B}$ and the constraint Ψ constitute the body of the rule. A goal has exactly the same format as the body of the rule of the form $? - \tilde{B}, \Psi$. If $\tilde{B}$ is an empty sequence of atoms, we call this a (constrained) fact. All goals, rules and facts are terms.*

The *universe of discourse* $\mathcal{D}$ of our CLP program is a set of terms, integers, and lists of integers. A *constraint* is written using a language of functions and relations. They are used in two ways, in the basic programming language to describe expressions and conditions, and in user assertions, defined below. In this report, we will not define the constraint language explicitly, but invent them on demand in accordance with our examples. Thus the terms of our CLP programs include the function symbols of the constraint language. A *ground instance* of a constraint, atom and rule is defined in obvious way. A *ground instance* of a constraint is obtained by instantiating variables therein from $\mathcal{D}$. The *ground instances* of a goal G, written $[\![G]\!]$ is the set of ground atoms obtained by taking all the true ground instances of G and then assembling the ground atoms therein into a set. We write $G_1 \models G_2$ to mean that for all groundings θ of G_1 and G_2, each ground atom in $G_1\theta$ appears in $G_2\theta$.

Let $G = (B_1, ..., B_n, \Psi)$ and P denote a goal and program respectively. Let $R = A :- C_1, ..., C_m, \Psi_1$ denote a rule in P, written so as none of its variables appear in G. Let $A = B$, where A and B are atoms, be shorthand for equations between their corresponding arguments. A *reduct* of G using R is of the form

$$(B_1, ..., B_{i-1}, C_1, ..., C_m, B_{i+1}, ..., B_n, B_i = A \wedge \Psi \wedge \Psi_1)$$

provided $B_i = A \wedge \Psi \wedge \Psi_1$ is satisfiable. A *derivation sequence* is a possibly infinite sequence of goals $G_0, G_1, ...$ where $G_i, i > 0$ is a reduct of G_{i-1}. If there is a last goal G_n with no atoms, notationally $(\Box, \Psi)$ and called a *terminal goal*, we say that the derivation is a *successful* and that the *answer constraint* is Ψ. A derivation is ground if every reduction therein is ground.

CLP has been successful as a programming language, and more recently, as a model of executable specifications. There have been numerous works which use CLP to model system modeling or programs and which use an adaptation of the CLP proof system for proving certain properties [16][6]. In this work, we follow this trends and use existing powerful constraint solvers for mechanized Timed Planning.

3 Timed Planning Specification

In our setting, Timed CSP is used to specify the timed tasks, which are multi-threaded and interactive. However, Timed CSP lacks expressiveness for stating system requirements which constraints all behavioral traces of a given process, i.e., the desired properties of Timed Planning. Thus, we extend Timed CSP with capability of stating requirements on process deadlines, event ordering, etc.

3.1 syntax

In a Timed Planning specification, a process is extended with an optional WHERE clause, which consists of a (first order) predicate over a predefined set of time variables. For instance, given a process P, the variable $P.\textsc{Start}$ ($P.\textsc{End}$) denotes the exact starting (ending) time of the process P. More specifically, $P.\textsc{Start}$ captures the starting time of a process P when its first event is enabled or when the "WAIT d" process is enabled if P starts with a WAIT process. $P.\textsc{End}$ is the ending time of the process P, i.e., the starting time of process STOP. If the process is non-terminating, then $P.\textsc{End} = \infty$. Naturally, $P.\textsc{End} \geq P.\textsc{Start}$ all the time. Using the two variables, a *deadline* property (a task must be accomplished within a certain time) is expresses as P WHERE $P.\textsc{End} - P.\textsc{Start} \leqslant d$ where d is a constant ($d \in \mathbb{R}^+$). In other scenarios, there may be a requirement on some event to occur at some exact time, e.g., attending a meeting at 10:00. A time variable ENGAGE is attached to an event e to denote the exact time when e is engaged, in the form of $e.\textsc{Engage}$.

Example 1. Kate needs to attend a meeting which starts at 10:00 and lasts less than two hours. She will go home after the meeting.

$$\begin{aligned} \mathit{Kate} \mathrel{\widehat{=}} & \mathit{arrive} \rightarrow \mathit{Meeting};\ \mathit{gohome} \rightarrow \textsc{Skip} \\ & \textsc{Where}\ \ \mathit{arrive}.\textsc{Engage} \leqslant 10 \wedge \mathit{Meeting}.\textsc{Start} = 10 \wedge \\ & \mathit{Meeting}.\textsc{End} - \mathit{Meeting}.\textsc{Start} \leqslant 2 \end{aligned}$$

where for simplicity we write 10 to represent 10:00. □

In addition, we use the variable $P.\textsc{Tes}$, where TES stands for *T*imed *E*vent *S*et, to record the engage time of all events engaged so far, which can be viewed as a history of the execution. We can retrieve the information we are interested in from the set TES. The following example illustrates a constraint concerning the number of occurrences of events in P.

Example 2. In a restaurant, the staff in the kitchen cook and supply food to the counter, the staff at the counter takes orders and delivers food to the customers.

$$\begin{aligned} \mathit{Kitchen} &\mathrel{\widehat{=}} \mathit{cook} \rightarrow \mathit{supply} \rightarrow \mathit{Kitchen} \\ \mathit{Counter} &\mathrel{\widehat{=}} \mathit{order} \xrightarrow{30} \mathit{serve} \rightarrow \mathit{Counter} \\ &\quad \textsc{Where}\ \ \mathit{serve}.\textsc{Engage} - \mathit{order}.\textsc{Engage} \leqslant 60 \\ \mathit{Mcd} &\mathrel{\widehat{=}} \mathit{Kitchen} \parallel \mathit{Counter} \\ &\quad \textsc{Where}\ \ \textsc{Tes} \downarrow \mathit{supply} \geq \textsc{Tes} \downarrow \mathit{serve} \wedge \textsc{Tes} \downarrow \mathit{supply} - \textsc{Tes} \downarrow \mathit{serve} \leqslant 10 \end{aligned}$$

$$
\begin{array}{lll}
\mathit{CP} & ::= \mathit{P}\ \textsc{Where}\ \ \mathit{WherePred} & \\
\mathit{WherePred} & ::= \mathit{WherePre} \wedge \mathit{WherePred} & \\
& \mid\ \mathit{WherePre} \vee \mathit{WherePred} & \\
& \mid\ \mathit{WherePre} \Leftrightarrow \mathit{WherePred} & \\
& \mid\ \mathit{WherePre} \Rightarrow \mathit{WherePred} & \\
& \mid\ \neg\, \mathit{WherePre} \mid \mathit{true} \mid \mathit{false} & \\
& \mid\ \mathit{WhereExpr} \sim \mathit{WhereExpr} & - \sim\ \in \{<, \leqslant, =, >, \geq\} \\
\mathit{WhereExpr} & ::= [\mathit{Name}.]\textsc{Start} & - \text{starting time of a process} \\
& \mid\ [\mathit{Name}.]\textsc{End} & - \text{ending time of a process} \\
& \mid\ [\mathit{Name}.]\textsc{Tes} & - \text{timed event set of a process} \\
& \mid\ \mathit{Name}.\textsc{Engage} & - \text{time of the occurrence of } \mathit{Name} \\
& \mid\ \mathit{Name}.\textsc{Engage}_i & - \text{time of the } i_{th} \text{ occurrence of } \mathit{Name} \\
& \mid\ \mathit{WhereExpr} \odot \mathit{WhereExpr} & - \odot\ \in \{+, -\}
\end{array}
$$

Fig. 1. Timed Planning Process Syntax

where $\textsc{Tes} \downarrow \mathit{supply}$ is the number of occurrences of event *supply* in the event set Tes. The Where clause guarantees for each order, there is available food to be delivered. □

The syntax of the Timed Planning process is summarized in Figure 1 where *Name* is a sequence of characters starting an alphabet, i.e., an event or a process. Note that if *Name* is missing, it defaults to the process name on the left hand side. To differentiate events of the same name in different processes, we write $P.a$ to denote the event a in P whenever necessary. By using the syntax defined above, common Timed Planning requirements can be specified easily.

We also add some functions over Tes to capture different aspects of the execution which can be used in the where predicates.

- $\textsc{Tes} \downarrow a$: the number of occurrences of event a in the timed event set Tes.
- *first*(Tes): The first event appear in Tes.
- *last*(Tes): The last event appear in Tes.

Example 3 (Common requirements). For instance, we can specify the *deadline* of a process, order of events, separation time between events and etc, which are shown in Table 1. □

3.2 Data types

The Timed Planning specification supports some primitive data types, including integers, real numbers, boolean values and list. We can define global variables

Requirements	TP representation
Process P must be finished within time d	P.END- P.START$\leqslant$ d
Process P must be finished before time d	P.END$\leqslant$ d
Max separation time between two events e_1, e_2 is d.	e_2.ENGAGE- e_1.ENGAGE$\leqslant$ d
e_2 must happen before e_1	e_2.ENGAGE- e_1.ENGAGE$\geq$ 0

Table 1. Basic Timed Planning Patterns

or process parameters.

$\mathbb{R}$: *RealNumber*
$\mathbb{N}$: *Integers*
$\mathbb{B}$: *BooleanType*
SEQ T : *Sequence of elements of type* T
SET T : *Set of elements of type* T

For SEQ T and SET T, type T $\in \{\mathbb{R}, \mathbb{N}, \mathbb{B}\}$.

3.3 Operational Semantics of Timed Planning

An operational semantics provides a way of interpreting a language by stepping through executions of programs written in that language. It describes an operational understanding of the language. The operational semantics of Timed CSP is precisely defined in Schneider [19] by using the combination of two relations: event transition and evolution. The semantics model for Timed Planning consists of three components: the event and timed transitions which are inherited from Timed CSP, a WHERE predicate which must be satisfied by this model, and an *Timed stamped set*. A *Timed stamped set* (Tss) is a record of an execution, consisting of a set of process related time stamps, namely the starting and ending times of processes, and a *timed event set* (TES) which is a set of timed events. TES is a subset of *Tss*: TES $\subseteq$ *Tss*. A *timed event* is a pair drawn from $e \times \mathbb{R}^+$ where $e \in \Sigma$, consisting of a time and an event engage time value.

We define the state of a process as a quadruple $\langle P, t, W, Tss \rangle$ where P is the process, t is the current time, W is the WHERE predicate and *Tss* is the *timed stamped set* of the model. *Tss* keeps value for all variables. At each transition, an evaluation of the system requirement W is performed. If the current state satisfies the requirement, the transition can be enabled, otherwise not.

Definition 3. *The operational semantics of a Timed Planning specification is a timed transition where the state is a quadruple* $\langle P, t, W, Tss \rangle$*, and event transitions and evolution transitions are defined by the rules:*

- $\langle P, t, W, Tss \rangle \xrightarrow{a} \langle P', t, W, Tss' \rangle$ *where* $\exists\, i : \mathbb{N} \bullet Tss \cup \{(a.\text{ENGAGE}_i, t)\} \vDash W$
- $\langle P, t, W, Tss \rangle \overset{d}{\rightsquigarrow} \langle P', t + d, W, Tss' \rangle$ *where* $d{>}0$

$$\frac{Tss \cup \{(e.\textsc{Engage}_i, t)\} \models W \qquad P_1 \xrightarrow{e} P_1'}{\langle P_1 \ {}_X||_Y \ P_2, t, W, Tss\rangle \xrightarrow{e} \langle P_1' \ {}_X||_Y \ P_2, t, W, Tss \cup \{(e.\textsc{Engage}_i, t)\}\rangle} [\ e \in X \cup \{\tau\} \setminus Y, Tes \downarrow e = i-1\ ]$$

$$\frac{Tss \cup \{(e.\textsc{Engage}_i, t)\} \models W \qquad P_2 \xrightarrow{e} P_2'}{\langle P_1 \ {}_X||_Y \ P_2, t, W, Tss\rangle \xrightarrow{e} \langle P_1 \ {}_X||_Y \ P_2', t, W, Tss \cup \{(e.\textsc{Engage}_i, t)\}\rangle} [\ e \in Y \cup \{\tau\} \setminus X, Tes \downarrow e = i-1\ ]$$

$$\frac{Tss \cup \{(e.\textsc{Engage}_i, t)\} \models W \qquad P_1 \xrightarrow{e} P_1' \qquad P_2 \xrightarrow{e} P_2'}{\langle P_1 \ {}_X||_Y \ P_2, t, W, Tss\rangle \xrightarrow{e} \langle P_1' \ {}_X||_Y \ P_2', t, W, Tss \cup \{(e.\textsc{Engage}_i, t)\}\rangle} [\ e \in X \cap Y, Tes \downarrow e = i-1\ ]$$

$$\frac{Tss \cup \{(\checkmark.\textsc{Engage}, t)\} \models W \qquad P_1 \xrightarrow{\checkmark} P_1' \qquad P_2 \xrightarrow{\checkmark} P_2'}{\langle P_1 \ {}_X||_Y \ P_2, t, W, Tss\rangle \xrightarrow{e} \langle P_1' \ {}_X||_Y \ P_2', t, W, Tss \cup \{(\textsc{End}, t)\}\rangle}$$

$$\frac{P_1 \overset{d}{\rightsquigarrow} P_1' \qquad P_2 \overset{d}{\rightsquigarrow} P_2'}{\langle P_1 \ {}_X||_Y \ P_2, t, W, Tss\rangle \overset{d}{\rightsquigarrow} \langle P_1' \ {}_X||_Y \ P_2', t+d, W, Tss\rangle}$$

Fig. 2. Operational Semantics of compositional operator $P_1 \ {}_X||_Y \ P_2$

where P' is the subsequent process of P by involving either a event transition ($\rightarrow$) or a timed transition ($\rightsquigarrow$). Tss' is a timed stamped set with updated Engage, Start, *and* End *variables.* □

The $\xrightarrow{a}$ represents an event transition, whereas $\overset{d}{\rightsquigarrow}$ is a timed transition. $\exists\, i : \mathbb{N} \bullet Tss \cup \{(a.\textsc{Engage}_i, t)\} \vDash W$ is an evaluation of the current state, which is used to check whether the current timed stamped set Tss fulfills the system requirements W or not.

In the operational semantics, we define both event transition and timed transition relations for all primary and compositional operators in Timed Planning specification, which are fully presented in [7]. The operational semantics of the alphabetized parallel composition operator $P_1 \ {}_X||_Y \ P_2$ are illustrated in Figure 2. The first two rules state that either of the components (P_1 or P_2) may engage an event as long as the event is not shared, only if the event with the current time satisfies the requirements. The next rule states that a shared event can be engaged simultaneously by both components as long as the event satisfies the requirements. The fourth rule is a special case for the third rule, whereas the event is the ✓ which is a special event used purely to denote termination. A new pair (End, t) is added to the Tss and hence checked. The last rule says that the composition may allow time elapsing when both the components do . We define the semantics rules for each operators in Timed CSP, which are fully illustrated

in [7].

Example 3 Take a printer process as an example. After the printer accepts a job, it needs to print this job within 30 to 60 seconds. Assume the process starts at time 0.

$$Printer \mathrel{\widehat{=}} accept \overset{30}{\rightarrow} print \rightarrow Printer$$
$$\text{WHERE} \quad print.\text{ENGAGE-}accept.\text{ENGAGE} \leqslant 60$$

W is the constraint $print.\text{ENGAGE-}accept.\text{ENGAGE} \leqslant 60$, which means $\forall\, i : \mathbb{N} \bullet print.\text{ENGAGE}_i\text{-}accept.\text{ENGAGE}_i \leqslant 60$. The following is one possible execution sequence.

$\langle accept \overset{30}{\rightarrow} print \rightarrow Printer, 0,$
$\quad \{print.\text{ENGAGE-}accept.\text{ENGAGE} \leqslant 60\}, \{(\text{START}, 0)\}\rangle \overset{10}{\rightsquigarrow}$
$\langle accept \overset{30}{\rightarrow} print \rightarrow Printer, 10,$
$\quad \{print.\text{ENGAGE-}accept.\text{ENGAGE} \leqslant 60\}, \{(\text{START}, 0)\}\rangle \overset{accept}{\rightarrow}$
$\langle \text{STOP} \triangleright \{30\}\ print \rightarrow Printer, 10, \{print.\text{ENGAGE-}accept.\text{ENGAGE} \leqslant 60\},$
$\quad \{(\text{START}, 0), (accept.\text{ENGAGE}_1, 10)\}\rangle \overset{30}{\rightsquigarrow}$
$\langle print \rightarrow Printer, 40, \{print.\text{ENGAGE} - accept.\text{ENGAGE} \leqslant 60\},$
$\quad \{(\text{START}, 0), (accept.\text{ENGAGE}_1, 10)\}\rangle \overset{20}{\rightsquigarrow}$
$\langle print \rightarrow Printer, 60, \{print.\text{ENGAGE} - accept.\text{ENGAGE} \leqslant 60\},$
$\quad \{(\text{START}, 0), (accept.\text{ENGAGE}_1, 10)\}\rangle \overset{print}{\rightarrow}$
$\langle Printer, 60, \{print.\text{ENGAGE} - accept.\text{ENGAGE} \leqslant 60\}$
$\quad \{(\text{START}, 0), (accept.\text{ENGAGE}_1, 10), (print.\text{ENGAGE}_1, 60)\}\rangle$
$\quad \ldots$

In this execution, event *accept* is firstly engaged at 10, we insert $(accept.\text{ENGAGE}_i 10)$ to Tss. It is not likely for event *print* to be firstly engaged after 40 seconds while it is enabled, where $print.\text{ENGAGE}_i$ will be greater than 70, since it is guarded by $print.\text{ENGAGE-}accept.\text{ENGAGE} \leqslant 60$. □

Healthiness Conditions As illustrated in the last section, each process is attached with a WHERE clause, which restricts the set of timed traces of the process. The constraints associated with a process are divided into two groups, namely the explicit ones defined in the WHERE clause and a set of implicit ones. The implicit constraints should always be true, i.e., a set of healthiness conditions. The following are two examples of such healthiness conditions. The complete list of healthiness conditions can be found in [7].

- For every event e in process P, e must be engaged between the starting time and ending time of P. Let αP be the alphabet of P.

$$\forall\, e : \alpha P \bullet P.\text{START} \leqslant e.\text{ENGAGE} \wedge e.\text{ENGAGE} \leqslant P.\text{END}$$

- Let P_i be a sub-process of P, written as $P_i \preccurlyeq P$. The starting time of P_i must be greater than or equal to the starting time of P and the its ending time must be less than or equal to that of P.

$$\forall\, P, P_i \bullet P_i \preccurlyeq P \Rightarrow P.\text{START} \leqslant P_i.\text{START} \wedge P_i.\text{END} \leqslant P.\text{END}$$

4 Verification of Timed Planning

In order to apply constraint solving technique to reason about systems modeled in the extended Timed CSP, we need to firstly encode the semantics of the processes as CLP rules. Once we encode the semantics of processes as CLP rules, well-established constraint solvers like CLP($\mathcal{R}$) [14] can be used to reason about those systems. Operational semantics defined in Section 3.3 are all encoded systematically.

The very initial step is to encode the syntax the extended Timed CSP into CLP. Note that this step can be automated by syntax rewriting. A relation $tproc(N, P, W)$ is used to present a process P of name N with where predicates W. W is a set of *wherePred* in a logical conjunction form. For instance, $W = [W1, W2, W3]$ means $wherePred = W1 \wedge W2 \wedge W3$. If there are no WHERE predicates defined for this process, W is the empty set. For instance, the syntax encoding of task *Kitchen* (Example 2) is as follows.

$tproc(kitchen, eventprefix(cook, eventprefix(supply, kitchen)), [\,]).$
$tproc(counter, delay(order, eventprefix(serve, counter), 30),$
$\quad [leq(engage(serve), engage(order))]).$
$tpoc(mc, parallel(kitchen, counter),$
$\quad [geq(number(tr(mc), supply), number(tr(mc), order)),$
$\quad\ leq(minus(number(tr(mc), supply), number(tr(mc), order)), 5)]).$

where relations '$leq, geq, minus, number$' are built-in predicates defined in our library, which represent '$\leqslant, \geq, -, \downarrow$' respectively.

Having defined the corresponding CLP syntax for the Timed Planning specifications, we devote the rest of this section to describe how the operational semantics are embedded as CLP rules. A relation of the form $tpos(P_1, T_1, E_1, M, P_2, T_2, E_2)$ is used to denote the *t*imed *p*lanning *o*perational *s*emantics, by capturing both event transition relations and evolution relations with a set of constraints. Informally speaking, $tpos(P_1, T_1, E_1, M, P_2, T_2, E_2)$ returns true if the process P_1 evolve to P_2 through either a time evolution, i.e., let $T_2 - T_1$ time units elapse (so that $M = []$), or an event transition by engaging an abstract event e instantly ($M = e$), as long as both transitions satisfy the WHERE requirements stored in E_1. After this transition relation, the local environment might change to E_2 by adding more predicates. E_1 (and E_2) is the environment of the system, which consists not only the WHERE predicates, but also the current values of the variables appeared in the WHERE predicates.

We define the $tpos/7$ [1] relation for each and every operator of the extended Timed CSP according to the semantics presented previously in Section 3.3.

$tpos(stop, T1, E, [], stop, T2, E) :- D >= 0, T2 = T1 + D.$
$tpos(skip, T, E1, [termination], stop, T, E2) :- sat(E, termination, T, E2).$
$tpos(skip, T1, E1, [], skip, T2, E2) :- D >= 0, T2 = T1 + D, sat(E1, T1, E2).$

[1] tpos/7 indicates the relation *tpos* with 7 parameters, same for sat/3 and sat/4.

The only transition for process STOP is time elapsing. Process SKIP may choose to wait some time before engaging the *termination* event which is our choice of representing ✓ in CLP. Process SKIP may not be able to terminate immediately since there might be some constraints involving P.END defined in the WHERE clause. The relation *sat* is required to be evaluated before the termination. Relation $sat(E_1, A, T, E_2)$ and $sat(E_1, T, E_2)$ are used to test whether the current state fulfills the requirements. Relation $sat/4$ handles event transition and $sat/3$ handles timed transition. The $sat/4$ and $sat/3$ rules are defines as:

$$
\begin{array}{l}
sat(E1, termination, T, E2) :- get\_process(E1, N), \\
\qquad insert(end(N, T), E1, E2), evaluate(E2). \\
sat(E1, A, T, E2) :- get\_process(E1, N), \\
\qquad insert(engage(A, N, T), E1, E2), evaluate(E2). \\
sat(E1, T, E1) :- evaluate(E1).
\end{array}
$$

The first rule says that whenever the *termination* event has been engaged, the predicate P.END $= T$ need to be validated in conjunction with all the current requirements stored in $E1$. Once it has been proved that the resultant predicate is not contradiction, we append this predicate to the current set of predicates. The second rule is to validate the case when an event is engaged, by adding the predicate A.ENGAGE $= T$ to the environment. The last rule captures the timed transition relation by evaluating the current environment with the current time. Relation $get\_process(E, N)$ is to find the current named process being executed. $evaluate(E)$ is to evaluate the current requirements, namely the constraint store. The detailed definition of $get\_process/2$ and $evaluate/1$ can be found in our web site[2].

In the operational semantics, there are a set of composition operators which are more complex than the primitive ones. For instance, the rules associated with the semantics of alphabetized parallel composition operator $P_1 \ {}_X||_Y \ P_2$ are as follows.

$$
\begin{array}{l}
tpos(para(P1, P2, X, Y), T, E1, A, para(P3, P2, X, Y), T, E2) \\
\quad :- member(A, X), not(member(A, Y)), \\
\qquad sat(E1, A, T, E2), tpos(P1, T, E1, A, P3, T, E2). \\
tpos(para(P1, P2, X, Y), T, E1, A, para(P1, P4, X, Y), T, E2) \\
\quad :- member(A, Y), not(member(A, X)), \\
\qquad sat(E1, A, T, E2), tpos(P2, T, E1, A, P4, T, E2). \\
tpos(para(P1, P2, X, Y), T, E1, A, para(P3, P4, X, Y), T, E2) \\
\quad :- member(E, X), member(E, Y), sat(E1, A, T, E2), \\
\qquad tpos(P1, T, E1, A, P3, T, E2), tpos(P2, T, E1, A, P4, T, E2).
\end{array}
$$

$$
\begin{array}{l}
tpos(para(P1, P2, X, Y), T1, E, [], para(P3, P4, X, Y), T1 + D, E) \\
\quad :- tpos(P1, T1, E, [], P3, T1 + D, E), tpos(P2, T1, E, [], P4, T1 + D, E).
\end{array}
$$

There is a one-to-one correspondence between our rules and the operators which are fully defined at [7]. This makes the soundness of our rules straightforward.

[2] `http://www.comp.nus.edu.sg/~zhangxi5/horae`

We have implemented a prototype reasoner based on one of the established CLP solver, namely CLP($\mathcal{R}$). CLP($\mathcal{R}$) is chosen for its support of real numbers and thus continuous time variables. A number of theories, libraries and shortcuts have been developed for easy querying and proving. This section is devoted to various proving we may perform over systems modeled using our extended Timed CSP. We discuss two main ones here. The first one is feasibility checking, which answers the schedulability problem. The other is safety and liveness checking.

4.1 Feasibility Checking

After specifying the tasks using the extended Timed CSP in CLP($\mathcal{R}$), the very first task is to check whether the tasks are feasible before simulation or reasoning of the system. Feasibility checking is necessary because there might be a conflict among the set of WHERE clauses of a system, which potentially invalidates any proving result. To perform this task, the conjunction of the WHERE predicates and the healthiness conditions are checked.

The output of the feasibility checking is either *yes* if the tasks are feasible or else *no*. In case the tasks are infeasible, i.e., there is no way to satisfy all the constraints, a minimum set of predicates which conflict each other can be generated so as to facilitate user correction easily. We use the CLP($\mathcal{R}$) predicate $feasibility\_checking(N, S)$ to fulfill this purpose, where N is the name of the process that is to be checked, and S is the minimum conflict set. If the process N is a feasible process $feasibility\_checking/2$ returns false, otherwise the minimum conflict set S is generated and returned.

$$
\begin{array}{l}
feasibility\_checking(N, S) :- get\_where(N, W), min\_cons\_store(W, S). \\
min\_cons\_store(W, S) :- find\_min(1, W, S). \\
find\_min(N, W, W) :- size(W, L), L < N, !. \\
find\_min(N, W, S) :- size(X, L), L >= N, delete\_at(N, W, WS), \\
\quad (satisfy(WS) -> find\_min(N + 1, W, S); find\_min(N, WS, S)).
\end{array}
$$

Relation $min\_cons\_store(W, S)$ is to generate the the minimum conflict set S of W if $W \models false$. It is performed by a linear scan on a sequence of constraints. We check whether after removing one constraint, the constraints store becomes satisfiable or not. If it does, then this constraint must be important and have to be put back, otherwise it can be discarded. It is an iterative process until a minimum set is found.

4.2 Reasoning about Safety and Liveness

Feasibility checking is to check whether the tasks modeled in Timed Planning are feasible. Once it is proven to be feasible, we can reason about safety or liveness properties by making explicit assertions.

Relation $reachable(P, Q, E1, E2, T1, T2, Tr)$ is defined to explore the full state space if necessary. Informally, it states that "process P starts at $T1$ with

environment $E1$ is able to be executed to Q at $T2$ with environment changed to $E2$ via trace Tr".

$$
\begin{array}{l}
reachable(P, P, \_, \_, T, T, []).\\
reachable(P, Q, E1, E2, T1, T2, N) :- tpos(P, T1, E1, A, P1, T3, E3),\\
\qquad (A = t(\_);\ A == tau;\ A = reccall(\_)), not\ table(P1),\\
\qquad assert(table(P1)), reachable(P1, Q, E3, E2, T3, T2, N).\\
reachable(P, Q, E1, E2\,T1, T2, [E \mid N]) :- tpos(P, T1, E1, A, P1, T3, E3),\\
\qquad not(A = t(\_);\ A == tau;\ A = reccall(\_)), not\ table(P1),\\
\qquad assert(table(P1)), reachable(P1, Q, E3, E2, T3, T2, N).
\end{array}
$$

reachable/7 is used to build assertions for various property checking. The first property of interest is to find one particular feasible schedule for the tasks, provided that the tasks are feasible. Relation $instance(P, Ins)$ is able to generate such feasible schedule Ins of process P.

$$
\begin{array}{l}
instance(P, Ins) :- not\ feasibility\_checking(N, \_), init\_Env(N, E),\\
\qquad reachable(P, \_, E, \_, 0, \_, Ins).
\end{array}
$$

where P specifies the tasks and Ins is a scheduling.

One property of special interest is deadlock-freeness. Relation $deadlock(P, Tr)$ is used to check the deadlock-freeness property, by trying to find a counterexample where P is deadlocked at some trace Tr.

$$
\begin{array}{l}
deadlock(P, Tr) :- init\_Env(P, E), reachable(P, P1, E, E2, 0, T2, Tr),\\
\qquad (tpos(P1, T2, E2, [t(\_)], Q1, T3, E3) \rightarrow\\
\qquad\quad not(tpos(Q1, T3, E3, A, \_, \_, \_), not\ A = [t(\_)]);\\
\qquad\quad not(tpos(P1, T2, E1, A, Q1, T3, \_), not\ A = [t(\_)])).
\end{array}
$$

It states that a process P at time 0 may result in deadlock if it can evolve to the process expression Q at time $T2$ where no event transition is available neither at $T2$ nor at any later moment. The last line outputs the trace which leads to a deadlock. Alternatively, we may present it as the result of the deadlock proving. Note that the above is different from the deadlock checking for standard Timed CSP as presented in [6]. Here the WHERE clauses at each step must be fulfilled. In general, a deadlock-free Timed CSP process may become a non deadlock-free process after it is enriched with certain WHERE clauses. It is, however, also possible for a non deadlock-free process to become deadlock-free.

We can also find the execution duration of an specific event, more specifically, the range of time that the event is able to be engaged. Relation $engage\_time(P, E, R)$ is defined for the purpose, which is to find the range R of the engage time of event E in process P. The detailed definition for all relations can be found in [7].

$$
\begin{array}{l}
engage\_time(P, E, []) :- not\ happen\_at(P, E, \_).\\
engage\_time(P, E, R) :- happen\_at(P, E, T), union(R, T, R1),\\
\qquad\qquad engage\_time(P, E, R1).
\end{array}
$$

where R is the range of engaged time of event E in process P which is generated after executing the relation. For instance,

$$P \mathrel{\widehat{=}} a \xrightarrow{5} b \xrightarrow{6} \textsc{Skip} \;||\; c \xrightarrow{7} b \xrightarrow{6} \textsc{Skip}$$
$$\textsc{Where}\;\; \textsc{End} - \textsc{Start} \leqslant 15 \wedge 3 \leqslant a.\textsc{Engage} \leqslant 7$$

The range of engage time of event b can be found by executing the following query $?- \mathit{engage\_time}(p, b, R)$. It returns result in the binding $R = [b\_\mathit{engage} \geq 8, b\_\mathit{engage} \leqslant 9]$, which indicates b.ENGAGE$\in$ [8, 9].

5 Case Study: The Pearl Harbor Attack

The attack on Pearl Harbor was a surprise military strike conducted by the Japanese navy against the United States' naval base at Pearl Harbor, Hawaii, on the morning of Sunday, December 7, 1941, later resulting in the United States becoming militarily involved in World War II. It was a carefully planned attack by the Japanese Navy which consisted of two aerial attack waves totaling 353 aircraft, launched from six Japanese aircraft carriers.

We studied some documents on plans and preparations of the attack from [9] and formalize the model of this attack using the Timed Planning specifications. The whole mission consists of several forces: air attack force, screening unit, support force, patrol unit, midway bombardment unit, reconnaissance unit and supply force, where each force has its own strength and duty.

Air Attack Force

> *" Air attacks will be carried out by launching the first attack units 230 nautical miles due north of Z point at 0130 hours (05:30am honolulu time) X Day (the day of the outbreak of hostilities), and the second attack unit at 200 nautical miles due north of Z point at 0245 hours....*
> **Carrier Striking Task Force Operation Order No.3** ***23 Nov 1941***"

The air force attack consists of two waves, we name take as *FirstAttack* and *SecondAttack*. According to their plan, the air force advanced their destination 20 hours before the attack. We model the air force attack as follows, where we use 0 to denote 00:00am such that 90 means 1:30 am and so on.

$$\begin{aligned}
\mathit{FirstAttack} &\mathrel{\widehat{=}} \mathit{launching}.24 \rightarrow \mathit{arrive.oahu} \rightarrow \mathit{attack} \rightarrow \mathit{return} \rightarrow \textsc{Skip} \\
&\quad \textsc{Where}\; \mathit{launching}.\textsc{Engage} = 90 \wedge 330 \leqslant \mathit{return}.\textsc{Engage} \leqslant 360 \\
\mathit{FirstAttack} &\mathrel{\widehat{=}} \mathit{launching}.24 \rightarrow \mathit{arrive.oahu} \rightarrow \mathit{attack} \rightarrow \mathit{return} \rightarrow \textsc{Skip} \\
&\quad \textsc{Where}\; \mathit{launching}.\textsc{Engage} = 165 \wedge 405 \leqslant \mathit{return}.\textsc{Engage} \leqslant 435 \\
\mathit{AirAttackForce} &\mathrel{\widehat{=}} \mathit{advance} \xrightarrow{1200} (\mathit{FirstAttack} \;|||\; \mathit{SecondAttack})
\end{aligned}$$

According to the document, for both attack, the aircrafts would return after 4 hours and within 4 and half hours of the start of each attack. *launching*.24 denotes the aircrafts are launching at speed of 24 knots.

The Midway Bombardment Unit The midway bombardment unit will depart from Tokyo Bay and after after refueling, secretly approach midway. It will arrive on the night of X Day and shell the air base. The unit will withdraw and, after refueling, return to the western part of the Inland Sea. We model the task of this unit as *MidwayBUnit*.

$$\begin{aligned} \mathit{MidwayBUnit} \mathrel{\widehat{=}} & \mathit{refuel} \rightarrow \mathit{depart} \rightarrow \mathit{arrive} \rightarrow \mathit{shell} \rightarrow \mathit{refuel} \rightarrow \mathit{return} \rightarrow \textsc{Skip} \\ & \textsc{Where}\ \mathit{arrive}.\textsc{Engage} \geqslant 1080 \end{aligned}$$

where constraint $\mathit{arrive.Engage} \geqslant 1080$ captures the idea that the unit would arrive at that night.

The Reconnaissance Unit This unit consists of different kind of reconnaissances.

- *Immediate Pre-attack Reconnaissance*: Two reconnaissance seaplanes will take off at 0030 hours, X Day, secretly reconnoiter Pearl Harbor and Lahaina Anchorage and report the presence of the enemy fleet.

- *Post-Attack Reconnaissance*: Before returning to the carrier, after the attack, an element of the fighters will fly as low as possible to observe and determine the extent of the damage inflicted upon the enemy aircraft and ships.

$$\begin{aligned} \mathit{IPAReconn} \mathrel{\widehat{=}} & \mathit{takeoff} \rightarrow (\mathit{launch.pearlHarbor} \rightarrow \mathit{reconnoiter} \rightarrow \mathit{report} \rightarrow \textsc{Skip} \\ & \quad ||| \ \mathit{launch.lahaina} \rightarrow \mathit{reconnoiter} \rightarrow \mathit{report} \rightarrow \textsc{Skip}); \\ & \quad \mathit{return} \rightarrow \textsc{Skip} \\ & \textsc{Where}\ \mathit{takeoff}.\textsc{Engage} = 30 \\ \mathit{PAReconn} \mathrel{\widehat{=}} & \mathit{takeoff} \rightarrow \mathit{observe} \rightarrow \mathit{report} \rightarrow \mathit{return} \rightarrow \textsc{Skip} \end{aligned}$$

$$\mathit{Reconnaissance} \mathrel{\widehat{=}} \mathit{IPAReconn} \ ||| \ \mathit{PAReconn}$$

The takeoff time of the Post-Attack Reconnaissance depends on the completeness of the main attack, so the constraint on the engage time of *takeoff* will be attached on a upper level process.

We model the task of each force as a process, the Pearl Harbor Attack on X Day is a composition of all sub tasks.

$$\begin{aligned} \mathit{PearlHarborAttack} \mathrel{\widehat{=}} & \mathit{AirForceAttack} \ ||| \ \mathit{MidwayBUnit} \ ||| \ \mathit{Reconnaissance} \\ & ||| \ \mathit{Supply} \ ||| \ \mathit{screening} \ ||| \ \mathit{Patrol} \\ & \textsc{Where}\ \mathit{PAReconn}.\textsc{Start} \leqslant \mathit{AirForceAttack}.\textsc{End} \end{aligned}$$

The documents recorded that in the initial plan, the Japanese Navy considered an assault plan in case the surprise attack does not succeed. But later they determined not to conduct this assault plan because at that time they had

complete confidence in the strength of the fighter units. If the assault plan is included in the whole plan, the model will be as follows:

$$\begin{aligned} \mathit{PearlHarborAttack2} \mathrel{\widehat{=}} & (\mathit{AirForceAttack} \mid\mid\mid \mathit{MidwayBUnit} \mid\mid\mid \mathit{Reconnaissance} \\ & \mid\mid\mid \mathit{Supply} \mid\mid\mid \mathit{screening} \mid\mid\mid \mathit{Patrol}) \\ & \triangledown \mathit{fail} \rightarrow \mathit{AssaultPlan} \\ & \text{WHERE } \mathit{PAReconn}.\text{START} \leqslant \mathit{AirForceAttack}.\text{END} \end{aligned}$$

6 Job-shop Scheduling

In this section, we show how the problem of job-shop scheduling can be modeled and solved using the Timed Planning specifications. We model both deterministic job-shop scheduling problem as well as the preemptive ones, which to our knowledge cannot be modeled and solved using purely Timed CSP specifications.

6.1 Job-Shop Scheduling Problem

The job-shop scheduling problem (JSSP) is a generic resource allocation problem in which common resources ("machines") are required at various time points (and for given durations) by different jobs. The goal is to find a way to allocate the resources such that all the jobs terminate as soon as possible, which is a schedule with the minimum time interval to finish all jobs. The difference between a deterministic and a preemptive job-shop scheduling problem is for the latter case, jobs can use a machine for some time, stop for a while and then resume from where they stopped. We define both problems formally as follows.

Definition 4. *(Job-shop scheduling problem) Given a set of $\mathcal{O}$ ol operations, a set $\mathcal{M}$ of* m *machines, and a set $\mathcal{J}$ of* n *jobs. For each operation $\nu \in \mathcal{O}$ there is a processing time $p(\nu) \in \mathbb{N}$, a unique machine $M(v) \in \mathcal{M}$ on which it requires processing, and a unique job $J(\nu) \in \mathcal{J}$ to which it belongs.* □

The difference between deterministic and preemptive job-shop scheduling problem is the feasible schedule.

Definition 5. *(Feasible Schedules for Deterministic Job-Shop Problem) A schedule is a function $S : \mathcal{O} \rightarrow \mathbb{N}$ that for each operation ν defines a start time $S(\nu)$. A schedule S is* feasible *if*

1. *Covering* :
   $\forall\, \nu \in \mathcal{O} : S(\nu) \geqslant 0,$
2. *Non-Preemption* :
   $\forall\, \nu, \omega \in \mathcal{O}, (\nu, \omega) \in A : S(\nu) + p(\nu) \leqslant S(\omega)$
3. *Mutual-Exclusion* :
   $\forall\, \nu, \omega \in \mathcal{O}, \nu \neq \omega, M(\nu) = M(\omega) :$
   $S(\nu) + p(\nu) \leqslant S(\omega)$ *or* $S(\omega) + p(\omega) \leqslant S(\nu).$

The problem is to find an optimal *schedule, i.e., a feasible schedule of minimum processing time.*

Definition 6. *(Feasible Schedules for Preemptive Job-Shop Problem) Let* $T(\mathcal{O}, i) \in \mathbb{N}$ *be processing time of the* i_{th} *step at which operation* $\mathcal{O}$ *executes. A schedule is a relation* $S \subseteq \mathcal{O} \times \mathbb{N} \times T$ *so that* $(\nu,\ st,\ t) \in S$ *indicates that operation* ν *starts to process on time* st *and processes for time* t*. A schedule* S *is* feasible *if*

1. *Ordering* :
   $\forall\, \nu, \omega \in \mathcal{O}, (\nu, \omega) \in A,$
   $(\nu, st, t) \in S, (\omega, st', t') \in S : st + t \leqslant st' + t'$
2. *Covering* :
   $\forall\, \nu \in \mathcal{O} : \sum_{(\nu,st,t)\in S} t = p(\nu)$
3. *Mutual-Exclusion* :
   $\forall\, \nu, \omega \in \mathcal{O}, \nu \neq \omega, (\nu, st, t) \in S, (\omega, st', t') \in S,$
   $M(\nu) = M(\omega) : st + t \leqslant st'\ \ or\ \ st' + t' \leqslant st.$

Consider M=$\{m_1$,$m_2\}$ and two jobs $J^1 = (m_2, 4)$ and $J^2 = (m_1, 3), (m_2, 4), (m_3, 6)$. The schedules S_1, S_2 and S_3 are depicted in Figure 3. The length of S_2 13 is the optimal schedule for a deterministic problem while S_3 which is 11 is the optimal schedule for a preemptive problem.

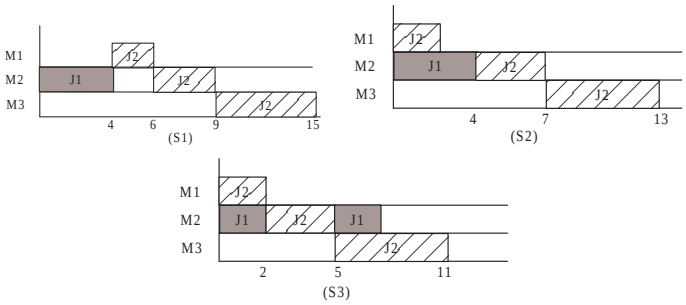

Fig. 3. The Gantt-Chart representations of three schedules

6.2 Modeling in Timed Planning

Deterministic Job-shop Scheduling Problem

Definition 7. *(Timed Planning for a Job) Let* $J^i = \langle o_1, ..., o_n \rangle$ *be a job, which is a chain of operations (ordered) on a set of machine* $\mathcal{M}$*. Its associated process presentation* P_i *is*

$$P_i \mathrel{\widehat{=}} m_1 \overset{p(o_1)}{\rightarrow} m_2 \overset{p(o_2)}{\rightarrow} ...m_k \overset{p(o_k)}{\rightarrow} \textsc{Skip}$$

where event m_j *in* P_i *denotes operation* o_j *of job* J^i *starts to process on machine* m_j *where* $m_j = \mathcal{M}(o_j)$*.* $p(o_j)$ *is the processing time required for* o_j *on* m_j*. Hence*

the delay process $m_j \overset{p(o_j)}{\rightarrow} m_{j+1} \overset{p(o_{j+1})}{\rightarrow}$ ... *denotes that* o_j *must process at machine* m_j *for* $p(o_j)$ *time units; then* o_{j+1} *can start to process which do not need to start immediately.*

Definition 8. *(Mutual Exclusion Constraints) Let* $\mathcal{J} = \{J^1, ..., J^n\}$ *be a job-shop specification and* $\mathcal{P} = \{P_1, ..., P_n\}$ *be a set of timed planning processes defined for each job.*

$$
\begin{array}{l}
\mathit{mutual\text{-}exclusion}(P) \mathrel{\widehat{=}} \\
\quad \forall\, P_i, P_j \in P, i \neq j, \forall\, m \in \sum P_i \cup \sum P_j \bullet \\
\qquad m \overset{t_i}{\rightarrow} P_i' \preccurlyeq P_i \wedge m \overset{t_j}{\rightarrow} P_j' \preccurlyeq P_j \Rightarrow \\
\qquad\quad P_i.m.\textsc{Engage} + t_i \leqslant P_j.m.\textsc{Engage} \vee \\
\qquad\quad P_j.m.\textsc{Engage} + t_j \leqslant P_j.m.\textsc{Engage}
\end{array}
$$

Definition 9. *(Timed Planning for Job-Shop specifications) Let* $\mathcal{J} = \{J^1, ..., J^n\}$ *be a job-shop specification and* $\mathcal{P} = \{P_1, ..., P_n\}$ *be a set of timed planning processes defined for each job. The timed planning presentation for the job-shop specification* $\mathcal{J}$ *will be an interleaving of all processes* P_i *with the non-delay and mutual exclusion constraints:*

$$
\mathit{JSSP} \mathrel{\widehat{=}} \; |||_{0<i\leqslant n} \;\; P_i \;\textsc{Where}\; \mathit{mutual\text{-}exclusion}(\mathit{JSSP})
$$

For every complete execution of JSSP (which terminates), its associated schedule S *is a* feasible *schedule. An optimal schedule is a trace of JSSP with the minimum ending time* $\mathit{min}(\mathit{JSSP}.\textsc{End})$.

Preemptive Job-shop Scheduling Problem

Definition 10. *(Timed Planning for a Preemptive Job) Let* $J^i = \langle o_1, ..., o_n \rangle$ *be a job, which is a chain of operations (ordered) on a set of machine* $\mathcal{M}$*. Each operation* o_i *should process on machine* m_j *for* $p(o_i)$ *time unit. Its associated process presentation for* o_i *is:*

$$
\begin{array}{l}
O_i \mathrel{\widehat{=}} \mu\, X \bullet m_s \rightarrow m_e \rightarrow \textsc{Skip};\; X \,\Box\, \textsc{Skip} \\
\quad \textsc{Where} \;\; \sum m_e.\textsc{Engage} - m_s.\textsc{Engage} \leqslant p(o_i) \wedge \\
\qquad\qquad \sum m_e.\textsc{Engage}\text{-}m_e.\textsc{Engage} = p(o_i) \Leftrightarrow \textsc{End} < \infty
\end{array}
$$

Event m_s *denotes operation* o_i *starts to process on its corresponding machine* m*,* m_e *denotes it leaves* m*. Expression* $\sum m_e.\textsc{Engage} - m_s.\textsc{Engage}$ *represents the total time* o_i *processes on* m*.*
The associated process presentation P_i *for each job is*

$$
P_i \mathrel{\widehat{=}} O_{i_1};\; O_{i_2};\; ...;\; O_{i_k}
$$

Definition 11. *(Mutual Exclusion Constraints) Let* $\mathcal{J} = \{J^1, ..., J^n\}$ *be a job-shop specification and* $\mathcal{P} = \{P_1, ..., P_n\}$ *be a set of timed planning processes defined for each job.*

$$
\begin{array}{l}
\mathit{mutual\text{-}exclusion}(P) \mathrel{\widehat{=}} \\
\quad \forall\, P_i, P_j \in P, i \neq j, \forall \{m_s, m_e\} \subseteq \sum P_i \cap \sum P_j \bullet \\
\qquad (\mu\, X \bullet m_s \rightarrow m_e \rightarrow ... \preccurlyeq P_i \wedge \\
\qquad \mu\, X \bullet m_s \rightarrow m_e \rightarrow ... \preccurlyeq P_j) \Rightarrow \\
\qquad\quad P_i.m_e.\textsc{Engage} \leqslant P_j.m_s.\textsc{Engage} \vee \\
\qquad\quad P_j.m_e.\textsc{Engage} \leqslant P_i.m_s.\textsc{Engage}
\end{array}
$$

Definition 12. *(Timed Planning for Preemptive Job-Shop specifications) Let $\mathcal{J} = \{J^1, ..., J^n\}$ be a job-shop specification and $\mathcal{P} = \{P_1, ..., P_n\}$ be a set of timed planning processes defined for each job. The timed planning presentation for the preemptive job-shop specification $\mathcal{J}$ will be an interleaving of all processes P_i with mutual exclusion constraints:*

$$
\begin{array}{l}
\mathit{PJSSP} \mathrel{\widehat{=}} |||_{0<i\leqslant n}\ \ P_i \\
\qquad \textsc{Where}\ \mathit{mutual\text{-}exclusion}(\mathit{PJSSP})
\end{array}
$$

6.3 Extended Job-shop Scheduling

Since Timed Planning is very flexible to specify system behaviors, with operational behaviors and critical timing constraints, those extended job shop scheduling problems with several additional features that are often specified in task scheduling problems can be easily and naturally modeled and solved using Timed Planning. In this project, we mainly focus on the compositional behaviors and the deadline and relative times extension. To our knowledge no other approaches are capable for those extensions.

Compositional Job Behaviors In traditional job-shop scheduling problems, all jobs are executed synchronously, which means that they are enabled at the same time. In our approach, jobs can be constructed in a more general way, for example, we can specify choices of jobs, sequences of jobs and interruption of one job by another job, by using the composition operators of Timed Planning. For example, a job-shop scheduling problem consists of 4 jobs $\mathcal{J} = \{J^1, J^2, J^3, J4\}$, where either J^1 or J^2 is running concurrently with J^3 which is interrupted by J^4 in 10 time units after execution. This scheduling problem can be specified as follows:

$$
\begin{array}{l}
\mathit{JSSP} \mathrel{\widehat{=}} (P1 \,\square\, P2) \,|||\, (P3 \,\triangledown\{10\}\, P4) \\
\qquad \textsc{Where}\ \mathit{mutual\text{-}exclusion}(\mathit{JSSP}) \wedge \mathit{non\text{-}delay}
\end{array}
$$

In our interpretation, jobs can communicate with each other through channels. A job $P1$ can activate another job $P2$ after some time t or after $P1$ finishes its first i_{th} operations. A job can also stop a job for some time units.

An extended job-shop scheduling problem consists of 2 jobs J^1, J^2, $J^1 = (1,2),(0,3),(2,5)$, $J^2 = (2,3),(0,3),(1,3)$. J^1 will activate J^2 after J^1 finishes its first operation. The Timed Planning model of this problem is specified as follows where a channel naming c is used to fulfill this purpose.

$$
\begin{aligned}
P1 &\ \widehat{=}\ 1 \xrightarrow{2} c!activate \rightarrow 0 \xrightarrow{3} 2 \xrightarrow{5} \textsc{Skip} \\
P2 &\ \widehat{=}\ c?activate \rightarrow 2 \xrightarrow{3} 0 \xrightarrow{3} 1 \xrightarrow{3} \textsc{Skip} \\
JSSP &\ \widehat{=}\ P1 \ |||\ P2 \\
&\quad \textsc{Where}\ mutual\text{-}exclusion(JSSP) \wedge non\text{-}delay
\end{aligned}
$$

Deadlines and Relative Timing Constraints Another extension of job-shop scheduling problem is that we can specify the deadline, relative timing constraints of each job. Those constraints can be naturally handled in Timed Planning by using its *Where* predicate. We formulate the constraints into 3 rules as follows:

Rule 1 Every operation o_i must be imperatively terminate before time $d(o_i)$
$\forall\, o_i \in \mathcal{O} \bullet S(o_i) + p(o_i) \leqslant d(o_i)$

Rule 2 Every jobJ^imust be terminate before time $d(J^i)$
$\forall\, J^i \in \mathcal{J} \bullet J^i.\textsc{End} \leqslant d(J^i)$

Rule 3 Relative timing constraints between operations
$\exists\, o_i, o_j \in \mathcal{O} \bullet p(o_i) \odot p(o_j) < t \quad , \odot \in \{+, -\}$

6.4 Experiments

We test ten problems among the bench marks of the job-shop scheduling problems on Windows XP with a 2.0 GHz Intel CPU and 1 GB memory. In Table 2 we compare our best result on the problems with the result reported in Table 15 of the survey paper [17], as well as the best result among randomly generated solutions for each problem.

7 Case Study: Timed Reminding System

Timed Reminding System is a specific application of the Timed Planning specification, where some features cannot be fully modeled using purely Timed CSP specification. For example, reminds the elderly to take medication at least 30 minutes after each meal, but cannot take more than once within 6 hours.

In this work, we provide a template of modeling timed reminders. Firstly, we classify reminders into four different types each of which has different models and parameters.

1. Timed-based Prompting
   - Timed fixed prompting services
   - Timed related prompting services (can be delayed within certain time)
2. Event-based Prompting
   - Event urgent prompting services: triggered by events and need to be prompt immediately.
   - Event related prompting services: triggered by events, but can be delayed up to an acceptable delay time, or must be delayed after an min_delay time.

Problem			Timed Planning			Random		Opt
name	♯j	♯m	time(s)	length	deviation	length	deviation	length
FT10	10	10	18	1001	7.63%	1761	89.35%	930
LA02	5	10	2	720	9.90%	1056	61.68%	655
LA19	10	10	0.2	902	7.12 %	1612	91.45%	842
LA21	15	10	102	1104	5.54%	2339	123.61%	1046
LA24	15	10	66	1007	7.58%	2100	124.00%	936
LA25	15	10	19	1098	12.38%	2209	126.10%	977
LA27	20	10	25	1441	16.68%	2809	127.45%	1235
LA29	20	10	112	1357	17.79%	2713	135.50%	1152
LA36	15	15	35	1341	5.57%	2967	133.90%	1268
LA37	15	15	56	1489	6.58%	3188	128.20%	1397

Table 2. The result for the ten hard bench marks deterministic job-shop scheduling problems. The first three columns give the problem name, no. of jobs and no. of machines. Out results (time in seconds, length of the best schedule and the deviation) appears next. The following two columns shows the best out of 2000 randomly-generated solutions, followed by the optimal result of each problem

7.1 Modeling and Specification

In this reminding system, locations of the participant are modeled as global variables which can be of type *boolean, integer, string or tuple*. We abstract the changes of the variables which should be updated by the environment. The activities of the participant, which should be detected by some sensing system, are modeled as *events*, such as *getup*, *leaving*, *take_medication*, *pick_phone* and etc.

Timed fixed reminders (TFR) Timed fixed reminders: The prompting is issued strongly at defined time so that the subject understands the urgency of executing certain activity. e.g.,

- get up at 7 a.m.
- catch flight at 11:30 a.m.
- go to attend lecture/meeting/seminar at 2p.m.

Timed fixed reminders can be modeled as processes whose starting time and prompting time must be exactly as the required time.

$$\begin{aligned} TFR(t) \mathrel{\widehat{=}} & [guard]prompt \rightarrow \textsc{Skip} \\ & \textsc{Where}\ calendar\_time(c, \textsc{Start}).time = st \wedge \\ & prompt.\textsc{Engage} = \textsc{Start} \end{aligned}$$

Example 4. The reminder which wakes up the elder at 7 a.m. would be:

$$\begin{aligned} wake\_up((7,0,0)) \mathrel{\widehat{=}} & [athome \wedge sleeping]prompt \rightarrow \textsc{Skip} \\ & \textsc{Where}\ calendar\_time(c, \textsc{Start}).time = (7,0,0) \wedge \\ & prompt.\textsc{Engage} = \textsc{Start} \end{aligned}$$

Timed related reminders (TRR) Timed related reminders: prompting is related to a time, but can be delayed within an acceptable time. e.g.,

- Start to cook dinner between 5 p.m and 5:30 p.m.
- Start to preparing tutorials/lectures at 3p.m.

At a specific time t, this reminder is enabled, but it do not need to prompt immediately if there are other reminders prompting. It need to be prompted within a acceptable delay d

$$\begin{aligned} TRR \mathrel{\widehat{=}} & [guard]prompt \rightarrow \textsc{Skip} \\ & \textsc{Where}\ calendar\_time(c, \textsc{Start}) \geqslant st \wedge \\ & \quad calendar\_time(c, promp.\textsc{Engage}) \leqslant st + dt \end{aligned}$$

Example 5. Remind the elderly to start preparing his dinner at around 5:30 p.m. to 5:30 p.m. every day if he/she is at home.

$$\begin{aligned} prep\_dinner((17,0,0),(17,30,0)) \mathrel{\widehat{=}} & [athome \wedge \neg sleeping]prompt \rightarrow \textsc{Skip} \\ & \textsc{Where}\ calendar\_time(c, \textsc{Start}).time \geqslant (17,0,0) \\ & \wedge\ prompt.\textsc{Engage} \leqslant (17,30,0) \end{aligned}$$

Event urgent reminders (EUR) Event urgent reminders: reminders triggered by event, which must be prompt as soon as the event is engaged.

- turnoff the stove after cooking
- bring the key while going out

Event urgent reminders can be modeled as processes whose first event is the triggering event.

$$\begin{aligned} EUR \mathrel{\widehat{=}} & trigger\_event \rightarrow [guard]prompt \rightarrow \textsc{Skip} \\ & \textsc{Where}\ trigger\_event.\textsc{Engage} = prompt.\textsc{Engage} \end{aligned}$$

Example 6. Remind the elderly to turnoff the stove after he finishes cooking.

$$\begin{aligned} turnoff\_stove \mathrel{\widehat{=}} & cooking\_end \rightarrow prompt \rightarrow \textsc{Skip} \\ & \textsc{Where}\ cooking\_end.\textsc{Engage} = prompt.\textsc{Engage} \end{aligned}$$

Event related reminders (ERR) Event related reminders: prompting is triggered by an event, but can be delayed within a accepted time, or must be delayed after a certain time. e.g.,

- wash hands after toilet within 3 minutes.
- take medication after 30 minutes of lunch.

$$\begin{aligned} ERR \mathrel{\widehat{=}} & trigger\_event \overset{mdt}{\rightarrow} [guard]prompt \rightarrow \textsc{Skip} \\ & \textsc{Where}\ \ prompt.\textsc{Engage} \leqslant trigger\_event.\textsc{Engage} + dt \end{aligned}$$

Example 7. Remind the elderly to wash hand after he finishes the toilet in 3 minutes.

$$\begin{aligned} Wash\_hand \mathrel{\widehat{=}} & finish\_toilet \rightarrow prompt \rightarrow \textsc{Skip} \\ & \textsc{Where}\ prompt.\textsc{Engage} \leqslant finish\_toilet.\textsc{Engage} + 3_{minute} \end{aligned}$$

Daily/ Weekly reminders The reminders that will prompt at a specific time every day, or every week with some conditions. For example, the wake up reminder which will prompt 8a.m. every weekdays. The attending lecture reminder which will prompt every 10a.m. every Monday if not public holidays.

Then we need other techniques (rules) to identify that whether today is weekdays or weekends or public holidays. We have two alternative options:

- get the information of whether today is weekdays or weekends from the environment by channels and abstract the details
- Maybe can use *Calendar Logic* to calculate those information in our system. In my opinion, Ontology can be a good candidate for modeling Calendar Logic.

Daily promoting services: prompt at time t everyday.

$$\begin{aligned} \mathit{Daily\_Reminder} \mathrel{\widehat{=}} & \mathit{Reminder};\ \mathit{Wait}\ 1_{\mathit{day}};\ \mathit{Daily\_Reminder} \\ & \textsc{Where}\ \mathit{calendar\_time}(c, \mathit{Reminder}.\textsc{Start}).\mathit{time} = t \end{aligned}$$

Daily promoting services: prompt at time t everyday, except Saturday and Sunday

$$\begin{aligned} \mathit{Daily\_Reminder} \mathrel{\widehat{=}} & \mathit{Reminder};\ \mathit{Wait}\ 1_{\mathit{day}};\ \mathit{Daily\_Reminder} \\ & \textsc{Where}\ 1 \leqslant \mathit{calendar\_time}(c, \mathit{Reminder}.\textsc{Start}).\mathit{week} \leqslant 5 \\ & \quad \wedge\ \mathit{calendar\_time}(c, \mathit{Reminder}.\textsc{Start}).\mathit{time} = t \end{aligned}$$

7.2 Priorities

In the Timed Reminding System, reminders have different priorities. Whenever more than one reminders are enabled at the same time, the one with the highest priority will be triggered and the others will be suspended.

In our settings, we introduce a new variable $e.\mathit{Pri}$ which records the priority value of event e. Whenever more than one event are enabled, if the priorities of all events are precisely specified, then the one with the highest priority is engaged. If only some of the events have priority, then all unspecified events and the one with the highest priority can be engaged.

We define both operational and denotational semantics of the priority.

7.3 Elderly Reminding System

The Elderly Reminding System is an instance of the Timed Planning System, which helps the dementias doing daily activities such as take medication at the right time, remind them prepare lunch, turn off stove after cooking and etc.

In this system, we consider the following reminders:

- wake up reminder (Timed fixed reminder)
- brush teeth (Timed related reminder)
- meal preparation (Timed related reminder)

– making phone calls (Timed related reminder)
– appointment (Timed related reminder)
– turn off stoves (Event urgent reminder)
– taking medication (Timed related reminder)
– bring keys while going out (Event urgent reminder)
– wash hands after toilet (Event related reminder)

Assumptions: the time stamp 00:00 a.m. is set to be 0. Hence 1:00 a.m. is 60, 2:00 a.m. is 120, and so on.

We need to define a set of global variables to keep the status of the elderly.

$athome : boolean$
$sleep : boolean$
$goingout : boolean$
$bringkey : boolean$
$washhand : boolean$

Wake up reminder Wake up the elderly at 7:00 a.m.

$$Wakeup \mathrel{\widehat{=}} [athome \wedge sleeping]prompt\_wakeup \rightarrow \textsc{Skip}$$
$$\textsc{Where } \textsc{Start} = calender\_time(c, \textsc{Start}) = (7, 0, 0)$$
$$\wedge\ prompt\_wakeup = \textsc{Start}$$

Watch TV Reminders the elderly to watch his favorite TV program at 10 am, which is timed fixed reminder.

$$TV \mathrel{\widehat{=}} [athome]prompt\_tv \rightarrow$$
$$\textsc{Where } calender\_time(c, \textsc{Start}) = (10, 0, 0) \wedge prompt\_tv.\textsc{Engage} = \textsc{Start}$$

Bring key Remind the elderly to bring his keys while going out, which is an event urgent reminder.
channel:{bring_key} is a sensor detecting whether the elderly brings his keys or not.
channel :{goingout}

$$Key \mathrel{\widehat{=}} goingout?yes \rightarrow ([\neg bringkey]prompt\_bringkey \rightarrow \textsc{Skip} \ \Box$$
$$[bringkey]\textsc{Skip})$$
$$\textsc{Where } prompt\_bringkey.\textsc{Engage} = goingout?yes.\textsc{Engage}$$

Wash hands Remind the elderly to wash hands after toilet, if the elderly does not wash his hand in one minute. It is an event related reminder.
chanel:{toilet, wash_hands}

$$WashHands \mathrel{\widehat{=}} toilet?finish \rightarrow (wash\_hand?yes \rightarrow \textsc{Skip} \ \triangledown\{1\}\ prompt\_washhand \rightarrow \textsc{Skip})$$
$$\textsc{Where } prompt\_washhand.\textsc{Engage} - toilet?finish.\textsc{Engage} \leqslant 5$$

Prepare Breakfast Remind the elderly to start preparing his breakfast at around 8:30 a.m. to 9:00 a.m. every day if he/she is at home.

$$\begin{aligned} \mathit{PrepareBreakfast} \mathrel{\widehat{=}} & [\mathit{athome} \wedge \neg \mathit{sleep}](\mathit{prompt\_breakfast} \overset{45_{minute}}{\rightarrow} \textsc{Skip}) \\ & \textsc{Where}\ \mathit{calender\_time}(c, \textsc{Start}) = (8, 30, 0) \wedge \\ & \qquad \mathit{prompt\_breakfast}.\textsc{Engage} \leqslant \textsc{Start} + 30 \end{aligned}$$

Medication Planner Basic system requirements: [20]

1. Never prompt outside the window: within a certain time interval: *[st, st+dt]*
2. Don't prompt if pill is already taken within the current window
3. Don't prompt if the participant is not at home: *athome=true*
4. Don't prompt if the participant is sleeping
5. Don't prompt if participant is on the phone
6. Prompting will resume if the participant returns home before the window expires
7. Prompt at plan 2 if participant is leaving
8. Wait till the time the user usually takes the pill. If it is earlier than the recommended pill taking time, start checking for plan 1 prompting opportunities at the usual pill time.
9. If only less than 20 minutes left till the window expires, start prompting at plan 1 disregarding all other rules (except 1-3)

Two kinds of prompting:

Plan 1 Prompt using the nearest device. The chime is played 10 seconds each time and lights stay on till location changes. Stop if pill is taken. Escalate to Plan 2 after 10 minutes.

Plan 2 Prompt using all prompting devices in the house every minute. Lights on devices stay on and chime is played for 10 seconds every minute.

$$\begin{aligned} \mathit{Plan}1 &\mathrel{\widehat{=}} \mathit{prompt} \overset{10}{\rightarrow} \textsc{Skip};\ \mathit{Wait}(10_{minute});\ \mathit{Plan}2 \triangledown \mathit{taken}?\mathit{yes} \rightarrow \textsc{Skip} \\ \mathit{Plan}2 &\mathrel{\widehat{=}} \mathit{prompt} \overset{10}{\rightarrow} \textsc{Skip};\ \mathit{Wait}(1_{minute});\ \mathit{Plan}2 \triangledown \mathit{taken}?\mathit{yes} \rightarrow \textsc{Skip} \end{aligned}$$

$$\begin{aligned} \mathit{Medi} \mathrel{\widehat{=}} (\ & [\mathit{athome} \wedge \neg \mathit{taken} \wedge \neg \mathit{onThePhone} \wedge \neg \mathit{sleeping}]\mathit{Plan}1 && \text{– rule 2 - 6} \\ & \quad \Box\ \mathit{leaving}?\mathit{yes} \rightarrow \mathit{Plan}2) && \text{– rule 7} \\ & \triangledown\{\mathit{dt} - 20_{minute}\}\ \mathit{Plan}2 && \text{– rule 8} \\ & \textsc{Where}\ \mathit{calendar\_time}(c, \textsc{Start}) \geqslant \mathit{st} \wedge \\ & \qquad \mathit{calendar\_time}(c, \mathit{promp}.\textsc{Engage}) \leqslant \mathit{st} + \mathit{dt} && \text{– rule 1} \end{aligned}$$

For rule 6 :Prompting will resume if the participant returns home before the window expires.
Our modeling is able to reserve this requirement in a more clever way. Once the participant returns home, the boolean variable *athome* will be changed to *true*, hence this process is able to be executed. So this model satisfies a more general requirement:

– Prompting will resume if the participant returns home or wakes up or finishes phone call before the window expires.

8 Conclusion

In this project, we formulated a more complicated scheduling problem, which we call Timed Planning. We have extended the Timed CSP with capabilities for stating planning related requirements in a compositional way. The Timed Planning has more expressive power than Timed Automata, for example it has the capability of keeping timed event set during execution and applying some operations on the set. Due to the expressiveness of CLP, it can express the syntax and semantics of Timed Planning fully. A mechanized proving system, based on CLP($\mathcal{R}$), has been developed to check the schedulability of a set of tasks and various safety and liveness properties can also be verified over systems modeled in Timed Planning. We studied the Japanese Navy plans for the Pearl Harbor Attack. According to the historical documents, we modeled the attack plan on December 7, 1941. We also applied Timed Planning to solve the job-shop scheduling problems which can be naturally modeled as Timed Planning processes. We also worked with the extended job-shop scheduling problems, where all jobs have composition operational behaviors. Besides, jobs with deadline and relative timing constrains are also able to be captured in our approach. We believe that the insight gained from this point of view will contribute both to scheduling and to the study of timed planning. We have demonstrated that the performance of the timed planning approach of solving job-shop scheduling problem can be highly improved by applying a set of optimizations. There are still many potential improvements to be explored to reduce the execution time, such as new partial-order methods and heuristics, etc.

Acknowledgements

We would like to thank Dr. Hiroshi Motoda for his very insightful comments and suggestions throughout this project. His high intellectual questions during our project presentations have helped us in many ways. We would also like to thank Dr. Terence Lyons, Dr. Bill Nace and Dr. Tae-Woo Park of AOARD/AFRL for many support for the past years. We would like to thank Dr. Paul B. Losiewicz of AFOSR/EOARD for suggesting our work to AOARD office in early years.

References

1. Yasmina Abdeddaïm and Oded Maler. Job-shop scheduling using timed automata. In *CAV '01: Proceedings of the 13th International Conference on Computer Aided Verification*, pages 478–492, London, UK, 2001. Springer-Verlag.
2. R. Alur and D. L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2), 1994.
3. P. Brucker. *Scheduling algorithms.* Springer, 1998.
4. J. Davies. *Specification and Proof in Real-Time CSP.* Cambridge University Press, 1993.
5. J. Davies and S. Schneider. A Brief History of Timed CSP. *Theor. Comput. Sci.*, 138(2).

6. J. S. Dong, P. Hao, J. Sun, and X. Zhang. A Reasoning Method for Timed CSP Based on Constraint Solving. In *ICFEM 2006*, pages 342–359, 2006.
7. J. S. Dong, J. Sun, and X. Zhang. Reasoning of Timed CSP and Extensions. In *Technical report*. http://www.comp.nus.edu.sg/∼zhangxi5/tp.pdf.
8. E. Fersman, P. Krcál, P. Pettersson, and Y. Wang. Task Automata: Schedulability, Decidability and Undecidability. *Information and Computation*, 205(8):1149–1172, 2007.
9. D. M. Goldstein and K. V. Dillon. *The Pearl Harbor Papers: Inside the Japanese Plans*. Potomac Books, 1999.
10. G.l Gupta and E. Pontelli. A Constraint-based Approach for Specification and Verification of Real-time Systems. In *IEEE Real-Time Systems Symposium*, pages 230–239, 1997.
11. C. A. R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice-Hall, 1985.
12. J. Jaffar and J. Lassez. Constraint Logic Programming. In *POPL*, pages 111–119, 1987.
13. J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
14. J. Jaffar, S. Michaylov, P. J. Stuckey, and R. H. C. Yap. The CLP(R) Language and System. *ACM Trans. Program. Lang. Syst.*, 14(3):339–395, 1992.
15. J. Jaffar, A. E. Santosa, and R. Voicu. A CLP Proof Method for Timed Automata. In *Real-Time Systems Symposium*, pages 175–186, 2004.
16. J. Jaffar, A. E. Santosa, and R. Voicu. Modeling Systems in CLP. In *Proceedings of the 21st International Conference on Logic Programming (ICLP 2005)*, volume 3668 of *Lecture Notes in Computer Science*, pages 412–413. Springer, 2005.
17. A. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present, and future, 1999.
18. S. Schneider. *Concurrent and Real-time System: The CSP Approach*. JOHN WILEY & SONS, LTD, 2000.
19. S. A. Schneider. An Operational Semantics for Timed CSP. In *Proceedings Chalmers Workshop on Concurrency, 1991*, pages 428–456. Report PMG-R63, Chalmers University of Technology and University of Göteborg, 1992.
20. Sengul Vurgun, Matthai Philipose, and Misha Pavel. A statistical reasoning system for medication prompting. In John Krumm, Gregory D. Abowd, Aruna Seneviratne, and Thomas Strang, editors, *Ubicomp*, volume 4717 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2007.

Appendix A: Operational Semantics of Timed Planning

Stop

$$\frac{}{\langle \textsc{Stop}, t, W, Tss\rangle \overset{d}{\rightsquigarrow} \langle \textsc{Stop}, t+d, W, Tss\rangle}$$

Event Prefix: a → Q

$$\frac{Tss \cup \{(a.\textsc{Engage}_i, t)\} \models W}{\langle a \rightarrow Q, t, W, Tss\rangle \overset{a}{\rightarrow} \langle Q, t, W, Tss \cup \{(a.\textsc{Engage}_i, t)\}\rangle}\ [\ Tes \downarrow a = i-1\ ]$$

$$\frac{}{\langle a \rightarrow Q, t, W, Tss\rangle \overset{a}{\rightsquigarrow} \langle Q, t+d, W, Tss\rangle}$$

Skip

$$\frac{Tss \cup \{(\checkmark.\textsc{Engage}, t)\} \models W}{\langle \textsc{Skip}, t, W, Tss\rangle \xrightarrow{\checkmark} \langle \textsc{Stop}, t, W, Tss \cup \{(End, t)\}\rangle}$$

$$\frac{}{\langle \textsc{Skip}, t, W, Tss\rangle \overset{d}{\rightsquigarrow} \langle \textsc{Skip}, t+d, W, Tss\rangle}$$

Timeout: $Q_1 \triangleright\{d\}\ Q2$

$$\frac{Tss \cup \{(a.\textsc{Engage}_i, t)\} \models W \qquad Q_1 \xrightarrow{a} Q_1'}{\langle Q_1 \triangleright\{d\}\ Q_2, t, W, Tss\rangle \xrightarrow{a} \langle Q_1', t, W, Tss \cup \{(a.\textsc{Engage}_i, t)\}\rangle}\ [\ Tes \downarrow a = i-1\ ]$$

$$\frac{Q_1 \xrightarrow{\tau} Q_1'}{\langle Q_1 \triangleright\{d\}\ Q_2, t, W, Tss\rangle \xrightarrow{\tau} \langle Q_1' \triangleright\{d\}\ Q_2, t, W, Tss\rangle}$$

$$\frac{}{\langle Q_1 \triangleright\{0\}\ Q_2, t, W, Tss\rangle \xrightarrow{\tau} \langle Q_2, t, W, Tss\rangle}$$

$$\frac{Q_1 \overset{d'}{\rightsquigarrow} Q_1'}{\langle Q_1 \triangleright\{0\}\ Q_2, t, W, Tss\rangle \overset{d'}{\rightsquigarrow} \langle Q_1' \triangleright\{d-d'\}\ Q_2, t+d', W, Tss\rangle}$$

External Choice: $Q_1 \,\Box\, Q_2$

$$\frac{Tss \cup \{a.\textsc{Engage}_i, t\} \models W \qquad Q_1 \xrightarrow{a} Q_1'}{\langle Q_1 \,\Box\, Q_2, t, W, Tss\rangle \xrightarrow{a} \langle Q_1' \,\Box\, Q_2, t, W, Tss \cup \{(a.\textsc{Engage}_i, t)\}\rangle}\ [\ Tes \downarrow a = i-1\ ]$$

$$\frac{Tss \cup \{a.\textsc{Engage}_i, t\} \models W \qquad Q_2 \xrightarrow{a} Q_2'}{\langle Q_1 \,\Box\, Q_2, t, W, Tss\rangle \xrightarrow{a} \langle Q_1 \,\Box\, Q_2', t, W, Tss \cup \{(a.\textsc{Engage}_i, t)\}\rangle}\ [\ Tes \downarrow a = i-1\ ]$$

$$\frac{Q_1 \xrightarrow{\tau} Q_1'}{\langle Q_1 \,\Box\, Q_2, t, W, Tss\rangle \xrightarrow{\tau} \langle Q_1' \,\Box\, Q_2, t, W, Tss\rangle}$$

$$\frac{Q_2 \xrightarrow{\tau} Q_2'}{\langle Q_1 \,\Box\, Q_2, t, W, Tss\rangle \xrightarrow{\tau} \langle Q_1 \,\Box\, Q_2', t, W, Tss\rangle}$$

$$\frac{Q_1 \overset{d}{\rightsquigarrow} Q_1' \qquad Q_2 \overset{d}{\rightsquigarrow} Q_2'}{\langle Q_1 \,\Box\, Q_2, t, W, Tss\rangle \overset{d}{\rightsquigarrow} \langle Q_1' \,\Box\, Q_2', t+d, W, Tss\rangle}$$

Parallel: $Q_1 \ {}_X||_Y \ Q_2$

$$\frac{Tss \cup \{(e.\textsc{Engage}_i, t)\} \models W \qquad Q_1 \xrightarrow{e} Q_1'}{\langle Q_1 \ {}_X||_Y \ Q_2, t, W, Tss\rangle \xrightarrow{e} \langle Q_1' \ {}_X||_Y \ Q_2, t, W, Tss \cup \{(e.\textsc{Engage}_i, t)\}\rangle} [\ e \in X \cup \{\tau\} \setminus Y, Tes \downarrow e = i - 1\ ]$$

$$\frac{Tss \cup \{(e.\textsc{Engage}_i, t)\} \models W \qquad Q_2 \xrightarrow{e} Q_2'}{\langle Q_1 \ {}_X||_Y \ Q_2, t, W, Tss\rangle \xrightarrow{e} \langle Q_1 \ {}_X||_Y \ Q_2', t, W, Tss \cup \{(e.\textsc{Engage}_i, t)\}\rangle} [\ e \in Y \cup \{\tau\} \setminus X, Tes \downarrow e = i - 1\ ]$$

$$\frac{Tss \cup \{(e.\textsc{Engage}_i, t)\} \models W \qquad Q_1 \xrightarrow{e} Q_1' \qquad Q_2 \xrightarrow{e} Q_2'}{\langle Q_1 \ {}_X||_Y \ Q_2, t, W, Tss\rangle \xrightarrow{e} \langle Q_1' \ {}_X||_Y \ Q_2', t, W, Tss \cup \{(e.\textsc{Engage}_i, t)\}\rangle} [\ e \in X \cap Y, Tes \downarrow e = i - 1\ ]$$

$$\frac{Tss \cup \{(\checkmark.\textsc{Engage}, t)\} \models W \qquad Q_1 \xrightarrow{\checkmark} Q_1' \qquad Q_2 \xrightarrow{\checkmark} Q_2'}{\langle Q_1 \ {}_X||_Y \ Q_2, t, W, Tss\rangle \xrightarrow{\checkmark} \langle Q_1' \ {}_X||_Y \ Q_2', t, W, Tss \cup \{(\textsc{End}, t)\}\rangle}$$

$$\frac{Q_1 \overset{d}{\rightsquigarrow} Q_1' \qquad Q_2 \overset{d}{\rightsquigarrow} Q_2'}{\langle Q_1 \ {}_X||_Y \ Q_2, t, W, Tss\rangle \overset{d}{\rightsquigarrow} \langle Q_1' \ {}_X||_Y \ Q_2', t + d, W, Tss\rangle}$$

Interleaving: $Q_1 \ ||| \ Q_2$

$$\frac{Tss \cup \{(e.\textsc{Engage}_i, t)\} \models W \qquad Q_1 \xrightarrow{e} Q_1'}{\langle Q_1 ||| \ Q_2, t, W, Tss\rangle \xrightarrow{e} \langle Q_1' ||| \ Q_2, t, W, Tss \cup \{(e.\textsc{Engage}_i, t)\}\rangle} [\ Tes \downarrow e = i - 1\ ]$$

$$\frac{Tss \cup \{(e.\textsc{Engage}_i, t)\} \models W \qquad Q_2 \xrightarrow{e} Q_2'}{\langle Q_1 ||| \ Q_2, t, W, Tss\rangle \xrightarrow{e} \langle Q_1 ||| \ Q_2', t, W, Tss \cup \{(e.\textsc{Engage}_i, t)\}\rangle} [\ Tes \downarrow e = i - 1\ ]$$

$$\frac{Tss \cup \{(\checkmark.\textsc{Engage}, t)\} \models W \qquad Q_1 \xrightarrow{\checkmark} Q_1' \qquad Q_2 \xrightarrow{\checkmark} Q_2'}{\langle Q_1 ||| \ Q_2, t, W, Tss\rangle \xrightarrow{\checkmark} \langle Q_1' ||| \ Q_2', t, W, Tss \cup \{(\textsc{End}, t)\}\rangle}$$

$$\frac{Q_1 \overset{d}{\rightsquigarrow} Q_1' \qquad Q_2 \overset{d}{\rightsquigarrow} Q_2'}{\langle Q_1 ||| \ Q_2, t, W, Tss\rangle \overset{d}{\rightsquigarrow} \langle Q_1' ||| \ Q_2', t + d, W, Tss\rangle}$$

Appendix B: Healthiness Conditions for Timed Planning

Implicit predicates for most of the process operators are defined as follows, where P denotes process, e denotes event, X and Y are set of events.

Event Prefix: $a^n \rightarrow P$

$$\forall\, a \rightarrow P \bullet a^n.\textsc{Engage} \leqslant P.\textsc{Start}$$

Sequence: $P1;\ P2$

$$\forall\, P1;\ P2 \bullet P1.\textsc{End} \leqslant P2.\textsc{Start}$$

Choice: $P1 \Box P2$

$$\forall\, P1 \mid\mid\mid P2 \bullet P1.\textsc{Start} = P2.\textsc{Start} \wedge P1.\textsc{End} = P2.\textsc{End}$$

Timeout: $P1 \triangleright \{d\}\ P2$

$$\forall\, P1 \triangleright \{d\}\ P2 \bullet init\{P1\}.\textsc{Engage} <= d \vee P2.\textsc{Start} = d$$

Interleaving: $P1 \mid\mid\mid P2$

$$\forall\, P_1 \mid\mid\mid P_2 \bullet P_1.\textsc{Start} = P_2.\textsc{Start} \wedge P_1.\textsc{End} = P_2.\textsc{End}$$

Interrupt: $P1 \bigtriangledown P2$

$$\forall\, P_1 \bigtriangledown P_2 \bullet P_1.\textsc{Start} \leqslant P_2.\textsc{Start}$$

Timed Interrupt: $P1 \bigtriangledown dP2$

$$\forall\, P_1 \bigtriangledown \{d\}\ P_2 \bullet P_1.\textsc{Start} + d \leqslant P_2.\textsc{Start}$$

Parallel: $P1\ {}_X||_Y\ P2$

$$\forall\, P_1\ {}_X||_Y\ P_2, \forall\, a \in X \cap Y \bullet P_1.\textsc{Start} = P_2.\textsc{Start} \wedge P_1.\textsc{End} = P_2.\textsc{End} \wedge P_1.a.\textsc{Engage} = P_2.a.\textsc{Engage}$$