Model-based Design for Real-time Software
Yan Li, Zhang Hong-jun

School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang, 212013, China

Summary
Model-based design is a novel approach for real-time embedded software. According to the demand of application in the data validity interval, real-time scheduling model directly can be derived based on the producer-consumer model. The precedence constraint problem in the producer-consumer model is solved by the mixed system of the time-triggered tasks and the event-triggered tasks. The model relaxes the deadline restriction for the task period, while preserving the synchronous computation semantics.

Key Words:
real-time scheduling; model-based design; precedence constraint; EDF algorithm; Schedulability

1. Introduction
The producer-consumer model is a common model in real-time systems. In the model the producer collects data, and the consumer carries out computation and application with these data. The producer-consumer model shall meet the precedence constraints[1]. In a hard real-time system, the real-time scheduling shall be predictable and all tasks shall be guaranteed to finish before the deadline.

Several approaches to the model-based design of real-time embedded software have been proposed in the literature. In[2], a method for automatic task construction was presented. In[3], a method for embedded software design from synchronous specifications was provided. In[4], a synchronous model for embedded software was studied. In[5], a mixed event-triggered and time-triggered system was outlined.

This paper presents a novel software model based on the demand of the data validity interval of the real-time data for embedded software design. We show how this mechanism can be implemented on single processor implementation platform.

2. System Model
Our model is based on multi-task system. The system uses the EDF (Earliest Deadline First) preemptive scheduling policy. The model assumes M producer tasks and Q consumer tasks coexisting in the system. The producer tasks are responsible for data generation and furnish consumers with data. The consumer uses the data furnished by the producer to carry out calculation and execute control operations or other operations.

A producer task $\tau_p$ corresponds to a consumer task set $\tau_c$. In $\tau_c$ there are n consumer tasks, i.e., $\tau_c = \{\tau_{c1}, \ldots, \tau_{cn}\}$.

The model assumes that the producer tasks and the consumer tasks are of one-to-many relationship. A consumer only relies upon a special producer. In the model, the producer task is a time-triggered task and the consumer task is a event-triggered task. A producer task $\tau_p$ gathers data periodically in a fixed interval $T_p$. $T_p$ is the period of $\tau_p$. After getting data, the producer task $\tau_p$ sends data to the buffer appointed by the system and, by event trigger activates all the consumer tasks in the consumer task set $\tau_c$. At this time the producer task $\tau_p$ has completed the work of this period and come to sleep, waiting for the re-activation by timer in the next period. After being activated, the consumer task reads data from the buffer for computation or control, and comes to sleep thereafter, waiting for the re-activation by the next event. The model neglects the time required by activation process.

The producer-consumer model constructed in this paper is shown in Fig. 1.

The task model is composed of a set of tasks, consisting of a virtually infinite sequence of jobs. The $k^{th}$ job of the task $\tau_i$ is denoted by $t_i(k)$. Every job is characterized by an arrival time $a_i(k)$, denoting the time at which the job is activation, a release time $r_i(k)$, denoting the time at which the job is put on the ready queue, and a start time $s_i(k)$ and a completion time $f_i(k)$, which denote the times at which the job starts and ends its execution. These quantities are characterized by the relations:

$$a_i(k) \leq r_i(k) \leq s_i(k) \leq f_i(k)$$

In this paper we only consider the situation wherein $a_i(k) = r_i(k)$.
The computation time and the worst-case computation time of the task \( \tau_i \) is denoted by \( C_i \) and \( WC_i \) respectively. The need to maintain consistency between the actual state of the environment and the state as reflected by the real-time data leads to the notion of temporal consistency. A real-time datum is defined as a triple: \( d(\text{value}, \text{time}, \text{avi}) \) \cite{6}, where \( d_{\text{value}} \) denotes the current value of \( d \); and \( d_{\text{time}} \) denotes the time when the observation relating to \( d \) was made, and \( d_{\text{avi}} \) denotes \( d \)'s absolute validity interval, i.e. the length of the time interval following \( d_{\text{time}} \) during which \( d \) is considered to have absolute validity. \( d_{\text{avi}} \) is also called the data validity interval.

For each consumer the data produced by the producer have a data validity interval. Different consumers may have different requirement on the time period of data, and they have different data validity intervals. The validity interval of real-time data is denoted by \( L \), i.e., \( d_{\text{avi}} \) as above. The consumer task shall finish its data processing within the data validity interval. The interval starting from data generation to the end of data validity interval is called data deadline, and the interval from the release of task to the completion of task is called task deadline.

\( L_{c_i} \) is defined as the data validity interval required by the consumer task \( \tau_{c_i} \). \( D_{c_i} \) is defined as the deadline of the consumer task \( \tau_{c_i} \). The data validity interval includes the time for the producer task \( \tau_p \) to acquire data and the time for the consumer task \( \tau_{c_i} \) to process data.

The real-time scheduling design of a system shall meet the following requirements:

\begin{itemize}
    \item The producer tasks and the consumer tasks shall satisfy the precedence constraints.
    \item The deadline \( D_{c_i} \) for the consumer task to data processing shall meet the requirement of the data validity interval \( L_{c_i} \).
\end{itemize}

The precedence constraint demands the consumers tasks to read data from the buffers for data processing after the producer task \( \tau_p \) has acquired data and sent them to the buffers. According to the definition of the model, the producer task \( \tau_p \) only generates the trigger to the consumer task after acquiring data, so the model has satisfied the precedence constraint.

The deadline for data processing of the consumer task \( \tau_{c_i} \) shall meet the requirement of the data validity interval. By formalization we express the deadline requirement as:

\[ \forall i \forall k \ (L_{c_i} \leq f_{c_i}(k) - r_p(k)) \quad (1) \]

where \( f_{c_i}(k) \) is the finish time for the consumer task \( \tau_{c_i} \), \( r_p(k) \) is the release time of the producer task \( \tau_p \).

Different consumer task may have different requirements of the data validity interval. We give the formula for calculating out the deadline \( D_{c_i} \) of the consumer tasks by the data validity interval \( L_{c_i} \) of the consumer task \( \tau_{c_i} \):

\[ D_{c_i} = (1 - \varepsilon) L_{c_i} \quad (2) \]

where \( \varepsilon \) is a relatively small positive real number, which is called relaxation factor, and makes the deadline \( D_{c_i} \) of the consumer tasks somewhat lower than the data validity interval \( L_{c_i} \).

Assume the earliest deadline of each task in a consumer task set \( \tau_c \) is \( D_{\text{min}} = \min(D_{c_i}) \). The period \( T_p \) of the producer task \( \tau_p \) shall be as following to ensure freshness of the data:

\[ T_p = D_{\text{min}} = \min(D_{c_i}) \quad (3) \]

The deadline \( D_p \) of the producer task \( \tau_p \) shall be the same as \( T_p \):

\[ D_p = T_p \quad (4) \]

The model adopts the EDF scheduling algorithm. We need to calculate out the current deadlines of all the tasks. Assume the first job of the task \( k = 1 \). At the moment \( t \) the job \( \tau_{p}(k) \) of the producer task \( \tau_p \) is being executed. By the following formula we can calculate the deadline \( D_p(k,t) \) of \( \tau_{p}(k) \) at the moment \( t \):

\[ D_p(k,t) = k T_p - t \quad (5) \]

where \( (k-1) T_p \leq t < k T_p \).

Obviously there shall be time reserved for the execution of the corresponding consumer task set \( \tau_c \) after the completion of the producer task \( \tau_p \). The consumer task at executing shall detect the reserved time. The job \( \tau_{c_i}(k) \) of the consumer task \( \tau_{c_i} \) has the deadline \( D_{c_i}(k,t) \) at the moment \( t \):

\[ D_{c_i}(k,t) = L_{c_i} + (k - 1) T_p - t \quad (6) \]

where \( f_p(k) \leq t \leq (k-1) T_p + L_{c_i} \).

**Theorem 1.** The deadline \( D_{c_i}(k,t) \) of \( \tau_{c_i}(k) \), calculated out from (5) and (6), meets the requirement of the data validity interval of the consumer task \( \tau_{c_i} \).

**Proof.** See Fig. 2, the producer task \( \tau_p \) corresponding to the consumer task \( \tau_{c_i}(k) \) is to be released from the start point \( (k-1) T_p = r_p(k) \) of the current period; the validity interval of the data consumed by \( \tau_{c_i} \) is calculated from this point. The producer task \( \tau_p(k) \) is to be finished on the moment \( f_p(k) \), triggering the release of the consumer task.
\( \tau_{ci}(k) \). To meet the data validity interval of \( \tau_{ci} \), time to be reserved for the execution of \( \tau_{ci}(k) \) at \( f_p(k) \) shall be:

\[
D_{ci}(k, f_p(k)) = L_{ci} - (f_p(k) - (k - 1) T_p)
\]  

(7)

This time is the deadline of \( \tau_{ci}(k) \) at \( f_p(k) \). Assume the current time to be \( t \), the deadline reduces due to the increase of \( t \), so (7) shall be

\[
D_{ci}(k, t) = L_{ci} - (t - (k - 1) T_p)
\]

(8)

\( \tau_{ci} \) + \( (k - 1) T_p - t \), i.e. (6).

From Theorem 1 we know that the consumer task has no need to be finished within the period of the producer task.

From (5) and (6) in the model to calculate out the processor demand of the producer task \( \tau_p \), and of the consumer task \( \tau_{ci} \) required by EDF algorithm.

In order to guarantee that the consumer tasks read correctly the data generated by the producer task from the buffers, because the deadline \( D_{ci} \) of the consumer task may be longer than the period of the producer task \( \tau_p \), the number \( b \) of the buffers required by the producer task \( \tau_p \) shall be calculated out by the following formula:

\[
b = \left\lceil \frac{L_{max}}{T_p} \right\rceil
\]

(8)

where \( L_{max} = \max(L_{ci}) \) is the maximum validity interval of the consumer tasks in the consumer task set \( \tau_c \) corresponding to the producer task \( \tau_p \).

The formula for the calculation of the buffer index is

\[
\text{Index}_{ci} = \text{Index}_p = k \mod b
\]

(9)

where \( k \) is the current job sequence number of \( \tau_p \) and \( \tau_{ci} \), separately, \( b \) is the number of the buffers, and the operator \( \mod \) is to get the residual.

Based on the model proposed in this paper, we can calculate out the basic parameters and dynamic parameters for real-time scheduling and complete the real-time scheduling design in accordance with basic requirements.

### 3. Real-time Scheduling Design

The real-time scheduling design given in this paper is completely model-based. From the model stated above we can see that we may realize the real-time task scheduling design by knowing the number of producer tasks, the consumer task sets corresponding to each producer task, and the demand of consumers on the data validity interval. By (2) to (4) in this model, we may calculate out the deadline \( D_{ci} \) of each consumer task, period \( T_p \) and deadline \( D_p \) of producer task \( \tau_p \). During system running we may by (5) and (6) in the model to calculate out the current deadlines of the producer task \( \tau_p \) and of the consumer task \( \tau_{ci} \) required by EDF algorithm.

4. Schedulability Analysis

In the model the deadline of the consumer task does not necessarily equal to the time period. The analysis of schedulability can be performed using a processor demand criterion [7].

The processor demand of a task \( \tau_i \) in any time interval \([t, t+L]\) is the amount of processing time required by \( \tau_i \) in \([t, t+L]\) that has to complete at or before \( t+L \).

For periodic tasks the processor demands all the tasks during the time interval \([0, L]\) can be expressed by

\[
\text{Index}_{ci} = \left\lceil \frac{L}{T_i} \right\rceil C_i
\]

(10)

**Proof.** By the schedulable condition of EDF[8], the processor utilization ratio is

\[
U = \sum_{i=1}^{N} \left\lceil \frac{L}{T_i} \right\rceil C_i \leq 1
\]

If \( U \leq 1 \), then for all \( L, L \geq 0 \),

\[
L \geq \sum_{i=1}^{N} \left\lceil \frac{L}{T_i} \right\rceil C_i
\]

(11)

Now assume \( U > 1 \) and prove that exist an \( L \geq 0 \) for which (10) does not hold. if \( U > 1 \), then For hyperperiod \( L = H = \text{LCM}(T_1, T_2, \ldots, T_N) \),

\[
L < \sum_{i=1}^{N} \left\lceil \frac{L}{T_i} \right\rceil C_i = \sum_{i=1}^{N} \left\lceil \frac{L}{T_i} \right\rceil C_i
\]

The processor demands of all tasks can not be satisfied during the time interval \([0, L]\).

**Theorem 2.** A set of periodic task is schedulable by EDF, if and only if for all \( L, L \geq 0 \),

\[
L \geq \sum_{i=1}^{N} \left\lceil \frac{L}{T_i} \right\rceil C_i
\]

(10)

Theorem 2 suffices to test equation (10) only for value of \( L \) equal to release times less than the hyperperiod \( H \). In
fact, if equation (10) holds for $L = r_k$, it will also hold any $L \in [r_k, r_{k+1}]$, since

$$\forall L \in [r_k, r_{k+1}], \quad \frac{L}{T_r} = \frac{r_k}{T_r}$$

The value of $L$ for which equation (10) has to be tested can still be reduced to the set of release times within the busy period. The busy period is the smallest interval $[0, L]$ in which the total processing time $W(L)$ can be computed as

$$W(L) = \sum_{i=1}^{n} \frac{L}{T_i} C_i$$

Thus, the busy period $B$ can be defined as

$$B = \min \{ L | L = W(L) \}$$

and computed by the algorithm shown as following:

```plaintext
busy_period
L = \sum_{i=1}^{n} C_i;
L' = W(L);
H = LCM\{T_1, T_2, \ldots, T_n\};
while (L' \neq L) and (L' \leq H){
L = L';
L' = W(L);
if (L' \leq H) B = L;
else B = INFINITY;
}
```

When the system is overloaded, the processor is always busy and the busy period is equal to infinite. On the other hand, if the system is not overloaded, the busy period coincides either with the beginning of an idle time or with the release of a periodic instance.

The model proposed by this paper has no definition for the period of consumer task. By the calculation of busy period $B$, the system is unschedulable if the busy period is infinite; otherwise the system is schedulable.

### 5. An Example and Simulation

Table 1 gives an example of producer-consumer model. In the table there are 3 producers tasks and 10 consumers tasks. The tasks are marked by pairs $(G, I)$, $G$ for the group wherein the task resides, $I$ for the number of the tasks in the group. A task group consists of a producer task and $n$ consumer tasks. The sequence number of producer task is 0, and the sequence numbers of the consumer tasks are marked from 1.

In the table column $C$ means the estimated computation time $C_i$ of the task. We may use the worst-case computation time $WC_i$. Column $L_C$ means the data validity interval required by consumer tasks. By (2) to (4) of the model we can calculate out the deadline $D_C$ of consumer tasks, and period $T_P$ and deadline $D_P$ of producer tasks.

<table>
<thead>
<tr>
<th>Table 1: Example of producer-consumer model</th>
</tr>
</thead>
<tbody>
<tr>
<td>producer tasks</td>
</tr>
<tr>
<td>$C$</td>
</tr>
<tr>
<td>(1, 0)</td>
</tr>
<tr>
<td>(1, 2)</td>
</tr>
<tr>
<td>(2, 0)</td>
</tr>
<tr>
<td>(2, 2)</td>
</tr>
<tr>
<td>(3, 0)</td>
</tr>
<tr>
<td>(3, 2)</td>
</tr>
<tr>
<td>(3, 4)</td>
</tr>
</tbody>
</table>

The hyperperiod for the system scheduling is 100 calculated out by the periods of the producer tasks. The simulation test can be finished by the execution of 100 time slices. Put parameters of Table 1 into the simulation program compiled by this model for calculation. During calculation process no unschedulable errors are reported. All tasks are shown completely executed by simulation.

Table 2 lists the time slices of tasks obtained from simulation.

<table>
<thead>
<tr>
<th>Table 2: The simulation of task execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
</tr>
<tr>
<td>(1, 0)</td>
</tr>
<tr>
<td>(2, 0)</td>
</tr>
<tr>
<td>(3, 0)</td>
</tr>
<tr>
<td>(1, 1)</td>
</tr>
<tr>
<td>(1, 2)</td>
</tr>
<tr>
<td>(1, 3)</td>
</tr>
<tr>
<td>(2, 1)</td>
</tr>
<tr>
<td>(2, 2)</td>
</tr>
<tr>
<td>(2, 3)</td>
</tr>
<tr>
<td>(3, 1)</td>
</tr>
<tr>
<td>(3, 2)</td>
</tr>
<tr>
<td>(3, 3)</td>
</tr>
<tr>
<td>(3, 4)</td>
</tr>
</tbody>
</table>

We can see clearly the distribution of execution time of each task during system scheduling in Table 2. In this case the processor utilization ratio is:

$$U = \frac{\sum_{i=1}^{n} C_i}{T_P} = \frac{2 + 1 + 2 + 3 + 4 + 3 + 4 + 5 + 4 + 4 + 5 + 7 + 6}{20 + 50 + 100} = 0.4 + 0.32 + 0.26 < 0.98 < 1$$
6. Conclusions

This paper presents a method for the real-time scheduling design of embedded software based on producer-consumer model. The designer shall deliver the estimated calculation time of the tasks and the data validity interval of the consumer tasks according to application demand. By the model we can calculate out other related parameters for real-time scheduling and verify the constraints.

The model proposed in this paper is simple and convenient for use, and improves the efficiency of the design of real-time embedded software. This paper carries out schedulability analysis upon the model, and simulates the task scheduling of the system by a program. By analyzing into the simulation results, we may further optimize the system design.

References


