RTA User's Guide

RTA: Response Time Analyzer — Revision 1.2

Juan A. de la Puente
Departamento de Ingenieria de Sistemas Telemáticos
Universidad Politécnica de Madrid

September 30, 1998

Revision history

1.1	1998-05-20	Initial version
1.2	1998-09-30	Task set file format enhanced to include information on resources used by tasks. Automatic computation of task priorities, priority ceilings for shared objects, and worst case blocking times for tasks

Contents

Chapter 1

Introduction

RTA is a software tool that performs response time analysis for real-time systems. The analysis is based on fixed-priority scheduling theory [1], and includes most of the techniques which are commonly known as *Rate-Monotonic Analysis* [5]. Before using RTA you should be acquainted with at least the basics of fixed priority analysis theory as described in e.g. [5] or [3].

RTA is portable. Is has been written in Ada 95 using only the standard libraries, and has no dependence on any underlying environment. It can thus be compiled and installed with any valid Ada 95 compiler. In order to keep the software simple and portable, only a plain text, command-line style user interface is provided. Future versions of RTA may include graphic edition or additional analysis tools that can be used with the basic tool.

RTA has been developed by Juan Antonio de la Puente at the Department of Telematic Systems Engineering of the Technical University of Madrid (DIT/UPM). Its development has been partially funded by CICYT, the Spanish Board for Scientific and Technical Research under project TIC96-0614.

Chapter 2

Running RTA

2.1 Getting started

You can run RTA from any text shell on any machine where it has been properly installed (see appendix A if you need to install it first.) Change to the `examples` directory directly under the `rta` directory, and type

```
rta example.tsf
```

The parameter `example.tsf` is the name of an example *task set file* (see 2.2 and 2.3 below.) A task set file includes a description of the real-time attributes of the tasks that make up the system to be analysed (their *profiles*). RTA assigns priorities to the tasks in deadline-monotonic order, and priority ceilings to all shared resources according to the priority ceiling protocol. It then computes the response times for all the tasks, and prints the results on your terminal as shown in figure 2.1.

```
Response time analysis for task set Sample
-----------------------------------------
Id Task      A PR  Period  Offset  Jitter    WCET   Block Deadline Response Sch
-- --------- - -- ------- ------- ------- ------- ------- -------- -------- ---
 1 Task_3    P  3  30.000   0.000   0.000   8.000   2.000   30.000   10.000 Yes
 2 Task_2    P  2  40.000   0.000   0.000   6.000   0.000   40.000   14.000 Yes
 3 Task_1    P  1  50.000   0.000   0.000  19.000   0.000   50.000   47.000 Yes

Priority ceilings for shared resources
--------------------------------------
Id Name      PR
-- --------- --
 1 Lock_1     3
 2 Lock_2     2

Total processor utilization :  79.67%
```

Figure 2.1: Output of the example run.

The `examples` directory includes further task set files, which you can also analyse with the RTA program. See the comments on the files for an explanation of their particular characteristics.

2.2 Tasks and task sets

Response time analysis is performed on *task sets*. A task set is a finite set of concurrently executing tasks with timing attributes. There is no limit, other than imposed by the amount of available storage, on the number of tasks a task set can have.

The attributes of each task are put together in its *task profile*. A task profile is a data structure with the following elements:

- The task *name*;
- Its *activation pattern*. The currently supported activation patterns are:
  - *Periodic*: the task is released at regular intervals of time.
  - *Sporadic*: the task is released at irregular instants, in response to an *event* which is signalled by some other task. There is a minimum separation between two consecutive releases of the same task.
  - *Interrupt*: the task is released at irregular instants, in response to external *events*, which are signalled by interrupts. There is a minimum separation between two consecutive releases of the same task.
- The task *priority*. Tasks are assumed to be scheduled according to their priorities, with the highest priority task executing first. Priorities are positive numbers, with the higher number denoting the higher priority.

  The current version of RTA requires that each task has a different priority.

  Priorities can be computed as part of the RTA execution (this is the default), or manually entered in the input task set file, when the `-p` option is given (see 2.5.)
- The activation *period* for periodic tasks, or the minimum *separation* between consecutive releases for sporadic tasks.
- The activation *offset*. This value is usually set to zero, except for periodic tasks that are released with an offset with respect to the initial time (see 3.1.)
- The worst case activation *jitter*. This value is mostly relevant to sporadic tasks that are activated by other tasks or by the reception of a message on a communication channel.
- The worst case *computation time* of the task over a single activation cycle.

  The computation time of a task must not be greater than its period.
- The worst case *blocking time* caused by priority inversion from lower priority tasks (see 3.1.)

  Blocking times can be computed as part of the RTA execution (this is the default), or can be entered in the task set file when the `-b` option is given (see 2.5.)

- The worst case *interference* caused by higher priority tasks (see 3.2.) This value is computed by the RTA tool as part of the response time computation.
- The task *deadline*, that is the time by which the task activity is due to end, relative to the release time.

  The deadline of a task can have any value, even if it is greater than the task period.
- The worst case *response time*, as computed by the RTA tool (see 3.2.)

The task set file also contains the attributes of the shared resources (protected by mutual exclusion *locks*) which are used by two or more tasks each. The attributes of a shared resource are put together in its *lock profile*. A lock profile is a data structure with the following elements:

- The lock *name* (usually the same as the shared resource it protects.)
- Its *priority ceiling*, which equals the priority of the highest priority task that can lock the resouurce (see 3.1). Priority ceilings are positive numbers, with the higher number denoting the higher priority.

  Priority ceilings can be computed as part of the RTA execution (this is the default), or manually entered in the input task set file, when the `-c` option is given (see 2.5.)

In order to properly compute resoruce priority ceilings and task blocking times, the task set file also includes a list of the shared resources used by each task.

The task set data, including the individual task are read by the RTA program from a *task set file*. For each task, at least the name, activation pattern, period, computation time, and deadline, must be specified. The rest of the values can be set to zero.

2.3 Format of task set files

Task set files are text files, containing characters in the ISO-8859 character set, which can be edited with any plain text editor. They can also be automatically generated by static analysis tools from source code or design models.[1]

It is recommended that task set files be given names ending with `".tsf"`, although this is not required by the tool.

Task set files may contain any number of blank characters or blank lines.

Comments start with `"--"` and run through the end of the line.

A task set file begins with a declaration of the task set name, and the number of tasks and locks that it contains. This is followed by a list of task profiles and a list of lock profiles. The file ends with an `end` statement.

```
task_set_file ::=
   task set set_name with task_number tasks
   [ and lock_number locks ]
```

[1] The current version of RTA does not include any such tools.

```
is
       {lock_profile}
       {task_profile}
end name ;
```

Notice the semicolon terminating the `end` statement.

There must be exactly *task*_number task profiles and, if provided, *lock*_number lock profiles.[2]

A lock profile includes the name of the resource, and an optional priority ceiling valuen in parentheses:

```
lock_profile ::=
   lock lock_name [(priority_number)];
```

Notice the semicolon at the end of the lock profile.

A task profile includes the name of the task, its activation pattern, and a list of parameters separated by commas in parentheses. This may be followed by a list of locks used by the task:

```
task_profile ::=
   task task_ name is activation_pattern (
      priority_ number,
      period_ time, offset_ time, jitter_ time,
      computation_ time, blocking_ time,
      deadline_ time, response_ time)
      [uses lock_name {, lock_name}];

activation_pattern ::=
   periodic | sporadic | interrupt |undefined
```

All the parameters must be present, even those valued zero. Notice also the semicolon at the end of the task profile.

Names are strings beginning with a letter followed by any number of alphanumeric, '_'. '-', or '.' characters. Case is not significant, either for reserved words or names. Thus, `task_1`, `Task_1`, and `TASK_1` are equivalent.

Number values are written as unsigned integer numbers with no intervening spaces, commas, underscores or any other characters.

Time values are entered as unsigned real numbers, optionnally including a dot as a decimal point character, but no spaces, commas, underscores or any other characters. There is no predefined time unit, i.e. time values may represent seconds, milliseconds, or any other time unit, provided it is used consistently for the whole task set. In most cases you will probably find practical to use milliseconds as the time unit.

Figure 2.2 shows the contents of the `example.tsf` task set file (the input to the sample run in 2.1.) You may find it useful to look at the other task set files included in the `examples` directory.

[2] If no lock number is provided, zero is assumed.

```
-- Sample task set
task set Sample with 3 tasks and 2 locks is
   -- locks
   lock Lock_1;
   lock Lock_2;
   -- tasks
   task Task_1 is periodic (0, 50, 0, 0, 19, 0, 0, 50, 0);
   task Task_2 is periodic (0, 40, 0, 0,  6, 0, 0, 40, 0)
      uses Lock_1 (2), Lock_2 (5);
   task Task_3 is periodic (0, 30, 0, 0,  8, 0, 0, 30, 0)
      uses Lock_1 (5);
end Sample;
```

Figure 2.2: Example task set file (`example.tsf`).

2.4 Output

The results of the response time analysis are printed on the standard output (usually the user's terminal, unless redirected to a file). The output format is shown on figure 2.1. The output produced by the RTA program includes the task set name, the task profile attributes for each task,[3] and an indication of the task schedulability.[4] The tasks are sorted by priority, unless otherwise specified by the `-n` option (see 2.5). The resource locks are also listed, together with thier respective priority ceiling values. Finally, the total processor utilization of the task set is printed.

The results are not output if an error has been found in the task set file, or the total processor utilisation is above 100%. In the latter case the response time computation algorithm does not converge (see 3.2), and the task set is not schedulable.

You can redirect the output of RTA to another file using the `-o` option (see 2.5) or the shell redirection mechanism, if there is one available.

2.5 Command line format

The full RTA comand line format is:

`rta [-`*flags*`] `*input_file*` [-s `*save_file*`] [-o `*output_file*`]`

The meaning of the command line arguments is as follows.

flags is a string including one or more of the following flag characters:

- `v` Verbose. If this flag is set, the program writes messages on the standard error file indicating progress in execution. This flag is normally used only for debugging purposes.
- `h` Help. The program prints a help message on the the standard error file.

[3] In the current version of RTA the interference value is not printed in order to save space.
[4] A task is schedulable if its response time is less than or equal to its deadline.

`p` Do not assign priorities to tasks, but instead use the priority values provided in the task set file.

`c` Do not compute priority ceilings for resource locks, but instead use the values provided in the task set file.

`c` Do not compute task blocking times, but instead use the values provided in the task set file.

`u` Update. The input task set file is replaced by a new file includig the time attributes (interference and response time) which have been computed as part of the analysis.

`n` Do not sort the task set file. By the default, both tasks and resource locks are sorted by priorities.

Notice that all the flags must be entered in a single string preceded by a hyphen. Only the `-s` and `-o` options are entered separately.

`input_file` is the name of a task set file which is taken as input for the analysis.

This argument is mandatory, except when the `v` or `h` are specified and no further arguments or flags are given. In this case, only the version number or, respectively, help message (see 2.6) is printed.

`-s` *`save_file`* Save the task set resulting from the analysis, including the computed priority, interference, and response time values, into a file named *save_file*.

No attempt is made to check whether the file already exists. If this is the case, the file is overwritten and the previous contents is lost.

`-o` *`output_file`* Redirect the output file to the specified file.

No attempt is made to check whether the file already exists. If this is the case, the file is overwritten and its previous contents is lost.

2.6 Messages

The RTA program may issue different kinds of messages to the user. All the messages are shown on the standard error file, which is usally set to the user's terminal, unless redirected.

There three kinds of messages: information, warning, and error messages. All of them have been designed in order to be easily understandable.

2.6.1 Information messages

Information messages are shown whenever the `v` flag is set, in order to show progress of execution through the different parts of the RTA program.

A help message is hown when the `-h` flag is set, or when there is an error in the command line syntax.

2.6.2 Warning messages

Warning messages are shown to inform the user of problems that are not fatal, so that analysis is still possible, although perhaps with degraded performance or incorrect results.

2.6.3 Error messages

Error messages are shown to inform the user that a problem has been found that makes analysis impossible to be performed.

Some error messages may require special actions. These are:

`Error: could not read input file` *name*
`Error: could not update file` *name*

`Error: could not save on file` *name*

`Error: could not write results`*name*

All of the above are input-output error messages that are output when there is some problem with the named files. Possible error causes are:

- Input errors: The file does not exist or is corrupt.
- Output errors: The file cannot be created or there is no available space on disk.

`Error: unknown error`
An unexpected error has occurred during RTA execution. When this happens, submit a bug report with all the required information.

Chapter 3

How RTA works

3.1 Computation model

The current version of response time analysis assumes a computation model consisting of a set of concurrent tasks running on a monoprocessor computer. The scheduling method is preemptive priority based [2].[1] All tasks must have a distinct integer priority, with higher values representing higher priorities.

Tasks may periodic or sporadic. Periodic tasks are released at times given by

$$t_r(k) = k \cdot T + O$$

or

$$t_r(k) = t_r(k-1) + T; t_r(0) = O$$

where T is the task *period* and O is the initial *offset*.

The actual time at which a task is activated may be delayed by the duration of a variable *jitter*, bounded by a maximum value J. The usal source of jitter for periodic tasks is *tick scheduling*. This is a common scheduling method which is based on a clock issuing ticks at regular intervals. The scheduler is activated on every clock tick in order to check which periodic tasks have become active. If the period of a task is not a multiple of the tick period, the task may suffer from jitter, with $J = T$. All the tasks that have periods which are multiples of the tick period have zero jitter.

Sporadic tasks are asociated to internal or external events, and are released at the times the event occurs. The release times of a sporadic task verify

$$t_r(k) \geq t_r(k-1) + T$$

In order to keep the system temporal behaviour predictable, a minimum *separation* between consecutive releases, T, is specified.

Sporadic tasks may also have jitter, which ususally comes from communication delays when a task is activated by other task by means of a message.

Task communication is assumed to be based on shared memory protected by *locks*. Critical sections must be executed according to some protocol that ensures

[1] See [3] for a detailed description of the principles and algorithms involved in response time analysis.

bounded blocking time, such as the *priority inheritance protocol* (PHP), the *priority ceiling protocol* (PCP), or the *immediate priority ceiling protocol* (IPCP), also known as the *highest locker protocol* (HLP) [8].[2] Conditional communication is limited to those cases in which the duration of waiting on a condition is bounded.

Worst case execution times (WCET) are assumed to be known for all the tasks. The maximum blocking times due to priority inversion on shared objects or conditional waiting can be computed in the following way when the PCP, IPCP or HLP protocol is used:

$$B_i = \max_{j \in LP(i), k \in HC(i)} C_{j,k} \tag{3.1}$$

where $LP(i)$ is the set of all tasks with priority lower than i, $HC(i)$ is the set of all locks with priority ceiling higher or equal than i, and $C_{j,k}$ is the computation time of the longest critical section executed by task j on lock k.

The WCET and blocking time of a task are denoted by C and B, respectively.

All tasks are assumed to have *deadlines*, denoted by D. A task is *schedulable* if its worst case response time is not greater than its deadline. Deadlines can be arbitrarily long.

3.2 Computing response times

The worst case response time of task τ_i, R_i, depends on three components:

- The worst case computation time of τ_i, C_i
- The maximum blocking suffered by τ_i, B_i
- The *interference* I_i, caused by preemption of τ_i by higher priority tasks.

The first two components are assumed to be known. The interference on τ_i over an interval $(0, t]$ is

$$I_i(t) = \sum_{j \in \mathrm{hp}(i)} \left\lceil \frac{t + J_j}{T_j} \right\rceil C_j \tag{3.2}$$

where $\mathrm{hp}(i)$ is the set of tasks with priorities higher than τ_i.

The response time is

$$R_i = (q+1)C_i + B_i + I_i(R_i) - qT_i + J_i \tag{3.3}$$

for periodic tasks, and

$$R_i = (q+1)C_i + B_i + I_i(R_i) - qT_i \tag{3.4}$$

for sporadic tasks, where $q+1$ is the number of releases of τ_i in the interval $(0, R_i]$.

Since both q and R_i are unknown, and appear on the rigt side of equations 3.3 and 3.4, an iterative algorithm is used to compute the response time. For $q = 0, 1, 2, \ldots$, the following recurrence relationship is applied:

[2] Examples of this kind of communication are Ada protected objects [7] and POSIX.1c mutexes [4].

$$w_i^{n+1}(q) = (q+1)C_i + B_i + \sum_{j \in \mathrm{hp}(i)} \left\lceil \frac{w_i^n + J_j}{T_j} \right\rceil C_j \tag{3.5}$$

A convenient starting value for $w_i^n(q)$ is

$$w_i^0(q) = (q+1)C_i + B_i \tag{3.6}$$

The iteration stops when a value $w_i^*(q)$ is found such that

$$w_i^{n+1}(q) = w_i^n(q) = w_i^*(q)$$

Then we have

$$R_i(q) = w_i^*(q) - qT_i + J_i \tag{3.7}$$

for periodic tasks, and

$$R_i(q) = w_i^*(q) - qT_i \tag{3.8}$$

for sporadic tasks.

The value of q is incremented until a value q^* is found such that

$$R_i(q^*) \leq T_i$$

The worst case response time of τ_i is then

$$R_i = \max_{q=0,1,\ldots,q^*} R_i(q) \tag{3.9}$$

For more details the reader is directed to [3].

3.3 Deadline monotonic priority assignment

It has been proved that for a large subset of the computation model, deadline-monotonic priorirty assignment is optimal [6]. The RTA tool assigns priorities to tasks in deadline monotonic order unless prevented by the `-p` option (see 2.5).

3.4 Features not yet supported

The effect of the run-time environment (e.g. context switch, interrupt handling, clock handler, etc.) is not included in the current version of the RTA tool.

Other features which will be added on subsequent releases of the RTA program include support for alternative locking protocols and priority assignment algorithms, as well as tasks with different criticality values.

Appendix A

Installing RTA

A.1 Retrieving RTA

The RTA installation files can be found on the DIT/UPM ftp server, at the URL:

`ftp://ftp.dit.upm.es/str/software/rta/`

The `pub/str/software/rta` directory at `ftp.dit.upm.es` contains several files:

- A README file with instructions for downloading and unpacking RTA distributions.
- A file called COPYING with a copy of the GNU Public License.
- Binary distribution files for unix platforms, containing `tar` archives compresed with `gzip`.[1]
- A binary distribution file for win32 platforms, containing a `tar` archive compressed with `gzip`.
- A source distribution file, containing a `tar` archive compressed with `gzip`.
- A documentation file containing copies of this guide in several formats.

Since RTA is written in Ada, you need an Ada 95 compiler in order to build RTA from the sources. There is a free compiler, GNAT, which can be dowloaded from `http://www.gnat.com` and many mirror sites.

Make sure that you download the files with ftp in binary mode.

A.2 Installing or building an RTA distribution

Intructions for installing and running RTA are in the README files which come with each distribution. The source distribution also contains instructions for building RTA.

[1] found at GNU servers.

Bibliography

[1] Neil Audsley, Alan Burns, Rob davis, Ken Tindell, and Andy J. Wellings. Fixed priority preemptive scheduling: An historical perspective. *Real-Time Systems*, 8(3):173–198, 1995.

[2] Alan Burns. Preemptive priority based scheduling: An appropriate engineering approach. In S.H. Son, editor, *Advances in Real-Time Systems*. Prentice-Hall, 1994.

[3] Alan Burns and Andy J. Wellings. *Real-Time Systems and Programming Languages*. Addison-Wesley, 2 edition, 1996.

[4] IEEE. *Threads Extensions for Portable Operating Systems*, 1995. IEEE 1003.1c.

[5] Mark H. Klein, Thomas Ralya, Bill Pollack, Ray Obenza, and Michael González-Harbour. *A Practitioner's Handbook for Real-Time Analysis. Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, Boston, 1993.

[6] J.Y.T. Leung and J. Whitehead. On the complexity of fixed-priority of periodic real-time tasks. *Performance Evaluation*, 2(4), 1982.

[7] International Standards Organisation. *Ada 95 Language Reference Manual*. ISO, 1995. International Standard ANSI/ISO/IEC-8652:1995.

[8] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Tr. on Computers*, 39(9), 1990.