Bound-T timing analysis tool

Using Bound-T in HRT Mode

Version 1
2005-04-06

Tidorum Ltd.

Tidorum Ltd
www.tidorum.fi
Tiirasaarentie 32
FI-00200 Helsinki
Finland

This document was written at Space Systems Finland Ltd by Niklas Holsti, Thomas Långbacka and Sami Saarinen.
The document is currently maintained at Tidorum Ltd by Niklas Holsti.

Document reference:	TR-UM-002
Document issue:	Version 1
Document issue date:	2005-04-06
Bound-T version:	2
Web location:	*http://www.bound-t.com/hrt-manual.pdf*

Credits:
This document was created with the free OpenOffice.org software (*http://www.openoffice.org/*).

The information in this document is believed to be complete and accurate when the document is issued. However, Tidorum Ltd. reserves the right to make future changes in the technical specifications of the product Bound-T described here. For the most recent version of this document, please refer to the web-site **http://www.tidorum.fi/**.

If you have comments or questions on this document or the product, they are welcome via electronic mail to the address **info@tidorum.fi**, or via telephone or ordinary mail to the address given below.

Please note that our office is located in the time-zone GMT + 2 hours, and office hours are 9:00 - 16:00 local time. In summer "daylight savings time" makes the local time equal GMT + 3 hours.

Cordially,

Tidorum Ltd.

Telephone: +358 (0) 40 563 9186
Web: http://www.tidorum.fi/
E-mail: info@tidorum.fi

Mail: Tiirasaarentie 32
FI-00200 Helsinki
Finland

Credits

The Bound-T tool was first developed by Space Systems Finland Ltd. (*http://www.ssf.fi/*) with support from the European Space Agency (ESA/ESTEC). Free software has played an important role; we are grateful to Ada Core Technology for the *Gnat* compiler, to William Pugh and his group at the University of Maryland for the *Omega* system, to Michel Berkelaar for the *lp-solve* program, to Mats Weber and EPFL-DI-LGL for Ada component libraries, and to Ted Dennison for the *OpenToken* package. Call-graphs and flow-graphs from Bound-T are displayed with the *dot* tool from AT&T Bell Laboratories.

This page is blank on purpose.

Contents

Tables

Figures

This page is blank on purpose.

1 INTRODUCTION

1.1 What Bound-T Is

Bound-T is a tool for developing real-time software - computer programs that must run fast enough, without fail. The main function of Bound-T is to compute an *upper bound* on the *worst-case execution time* (WCET) of a program or subprogram.

The function, "bound time", inspired the name "Bound-T", pronounced as "bounty" or "bound-tee".

The basic functions and usage of Bound-T are described in the Bound-T User Manual (reference [1]).

1.2 Hard Real Time Programming Model

Bound-T contains special high-level support for the programs that follow the *Hard-Real-Time (HRT)* programming model, an architectural style for concurrent, real-time programs originally defined by the European Space Agency.

The HRT methodology consists of

- a formalism for *modelling* real-time programs, and
- a method for *analysing* such models and determine if they are *schedulable*, that is, if all deadlines can be met.

HRT schedulability analysis needs upper bounds on the WCET of the executable objects in the HRT program, which is where Bound-T steps in. For an HRT program, Bound-T can generate so-called *execution skeletons* with detailed WCET information as required by HRT schedulability analysis.

This manual explains the "HRT mode" of Bound-T: what it is and how to use it. We assume that the reader is already familiar with the basic functions of Bound-T as explained in the User Manual [1]. The HRT mode is an extension of the basic Bound-T functions. The HRT mode is activated by a Bound-T command that contains the *-hrt* option. Without the *-hrt* option we say that Bound-T is used in the "basic mode".

1.3 Overview of this Manual

What the reader should know

We assume that the reader is familiar with the Bound-T User Manual [1] and the background knowledge it requires.

This manual will focus on the HRT modelling formalism and how to use Bound-T for WCET analysis of HRT programs. The schedulability analysis is not addressed. It is usually an application of deadline-monotonic, fixed-priority, preemptive scheduling with priority ceiling inheritance.

The manual is organised into chapters as follows:

- Chapter 2 introduces the HRT model, including the concepts of thread, resource object and synchronisation object.
- Chapter 3 explains how to execute Bound-T in HRT mode.
- Chapter 4 gives the syntax of the Threads and Protected Objects File (TPOF), which describes the structure of an HRT program and tells Bound-T which subprograms should be analysed in HRT mode.
- Chapter 5 gives the syntax of the Execution Skeleton File (ESF), which is the main output from Bound-T in HRT mode. The ESF augments the TPOF with the worst-case execution-time bounds and worst-case execution path that Bound-T has found.
- HRT-related warning messages and error messages are listed in Chapter 6, with explanations and advice on solving the problems.
- A glossary of HRT-related terms in Chapter 7 concludes the manual.

1.4 References

[1] Bound-T User Manual.
Tidorum Ltd., Doc.ref. TR-UM-001.
http://www.bound-t.com/user-manual.pdf

2 THE HRT MODEL

2.1 Introduction

The HRT (Hard Real Time) methodology consists of

- a formalism for modelling real-time programs, and
- a method for analysing such models and determine if they are schedulable, that is, if all deadlines can be met.

In the HRT model, a real-time program contains a fixed set of *threads* and a fixed set of *protected objects*. A thread is an active, executing entity (task, process) while a protected object is a passive, data entity, not far removed from a monitor. The threads interact only by calling *entries* in protected objects.

The HRT model is quite similar to the "Ravenscar profile" for Ada programs, but is not dependent on programming language or real-time kernel.

The Bound-T user may have to map the HRT concepts to the primitives of the particular programming language and real-time kernel used by the target program. Therefore, we describe the concepts of thread and protected object in some detail.

2.2 Threads

Root subprograms

A thread is a sequential process, executing concurrently (in the logical sense, at least) with other threads. In the program, a thread has a *root subprogram* representing the top level of the sequential algorithm. The root subprogram usually contains an endless loop of the form

```
loop
  wait for next activation signal or activation perod;
  perform one activation of the thread;
end loop;
```

The root subprogram can call other subprograms, so the control flow of the thread is by no means confined to the root subprogram.

In some real-time kernels a loop as above is not used, or is used only for background threads. Instead, the kernel invokes the root subprogram for each activation of the thread, and the root subprogram returns to the kernel when the activation is completed. This is often the case for threads activated by interrupts (interrupt handlers).

Cyclic or sporadic

A thread is either a cyclic thread, an interrupt-sporadic thread or a software-sporadic thread:

- A *cyclic* thread is activated periodically with a constant period particular to the thread.
- A *software-sporadic* thread is activated by some other thread(s) when needed, but with some minimum time interval between activations, which again is particular to the thread.

- An *interrupt-sporadic* thread is activated by a hardware interrupt, again with some minimum interval.

Once a thread is activated, it performs some computation (perhaps including input and output) and then suspends itself to wait for the next activation. As part of the computation, the thread may call entries of protected objects.

For any thread, the HRT model assumes knowledge of the worst-case execution time (WCET) of one activation, and also of the protected entries called by the thread.

From the programming language point of view, an Ada task corresponds exactly to an HRT thread, but in HRT the inter-thread communication is restricted and not all Ada facilities are allowed. POSIX threads can also be seen as HRT threads.

2.3 Protected Objects

A protected object is a means for controlled interaction between threads, and is either a *resource* object or a synchronisation object.

Resource objects

A resource object provides a set of entries (services, subprograms) that let two or more threads access some shared data in mutual exclusion. That is, while a thread is executing an entry of a protected object, no other thread can access the same protected object.

Synchronisation objects

A synchronisation object provides the means to activate a sporadic thread. It provides two entries: a "wait for activation" entry for use by the sporadic thread, and an "activate" entry for use by the activating threads.

Each synchronisation object is statically associated with a particular software-sporadic thread, and vice versa. In other words, a given software-sporadic thread always waits on its "own", fixed synchronisation object, and each synchronisation object is "owned" by some software-sporadic thread.

A protected entry (i.e. an entry in a protected object) can call other protected entries as long as the called entries are not blocking (have no barriers).

For each protected entry, the HRT model assumes knowledge of the WCET of the entry and which other protected entries it calls.

From the programming language point of view, an Ada protected object corresponds to an HRT protected object. However, as a matter of terminology, note that only the "wait for activation" entry of an HRT resource object corresponds to an Ada entry; other HRT "entries" correspond to protected subprograms in Ada. Furthermore, HRT allows only a subset of the Ada facilities.

POSIX mutex and semaphore objects can also be used as HRT resource objects and synchronisation objects, respectively. Again, the HRT model places restrictions on how these POSIX objects can be used.

2.4 An Example HRT program

To illustrate the HRT concepts and the Bound-T HRT mode, consider a program *Monitor_Example* that contains three threads:

- The *Sampler* thread runs cyclically at 100 Hz and reads sensor samples. It can raise an alarm if the value of a sample is above a threshold.
- The *Monitor* thread runs cyclically at 1 Hz to monitor the average value of the samples. It can raise an alarm if the average value is below a threshold.
- The *Reporter* thread is a sporadic thread that is activated when an alarm is raised, and is responsible for handling (reporting) the alarm.

These threads interact via two protected objects as follows:

- The protected object *Averager* is a resource object that provides two entries, *Averager_Add* to add a sample to a cumulative sum and count, and *Averager_Get* to return the average and reset the cumulative sum and count.
- The protected object *Alarmer* is a synchronization object that provides two entries, *Alarmer_Signal* to raise an alarm and store an alarm value, and *Alarmer_Wait* to wait until an alarm is raised and return the alarm value.

The following figure illustrates the interaction between the threads and protected objects in this example.

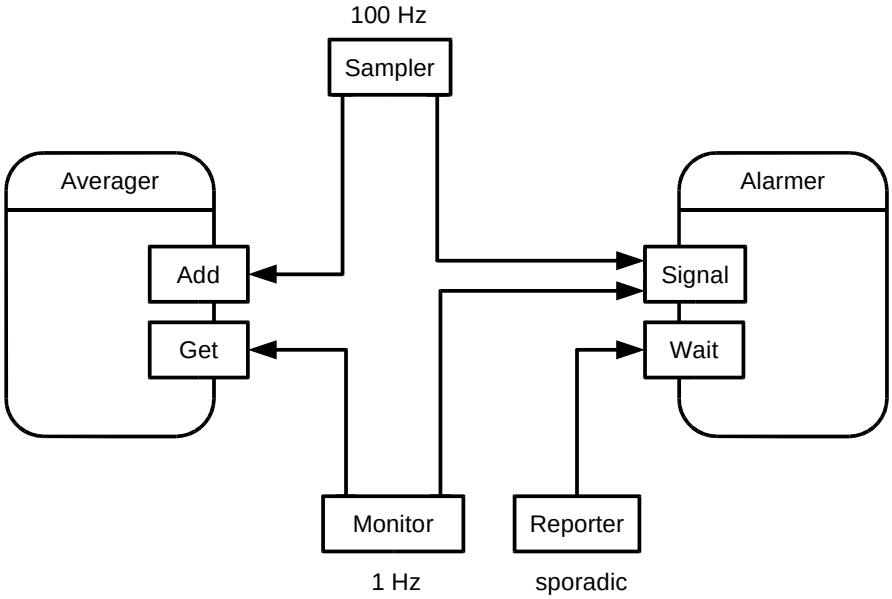

Figure 1: Structure of the Example HRT Program

2.5 The Threads and Protected Objects File

When Bound-T is used in the HRT mode to analyse a target program, in addition to the target program itself the user provides an input file that lists the threads and protected objects in the HRT structure of the target program. This file is called the *TPO file* (Threads and Protected Objects file) or *TPOF*.

The TPOF is a text file written in a specific syntax which is defined in Chapter 4 but which we hope is rather self-explanatory, given the concepts of the HRT model.

Example TPOF

A TPOF for the *Monitor_Example* program could be as follows; we have complicated the *Sampler* thread by defining two modes for it. (The underscores prefixed to thread and entry names compensate for underscores added by the compiler before linking; they are not a specific feature of the TPOF or Bound-T.)

```
program Monitor_Example

--   Monitor: Tiny HRT Example for Bound-T
--
-- This program has three threads and two protected objects,
-- one resource object and one synchro object.


protected Averager
--
-- The Averager object accumulates data samples into a sum
-- and counts. Then, the average value can be read out,
-- which also resets the sum and count.
--

   type resource
   entry _Averager_Add
   entry _Averager_Get

end Averager


protected Alarmer
--
-- The Alarmer object provides a one-place buffer for an
-- alarm value and Wait and Signal operations to wait for
-- and signal an alarm.

   type synchro

   entry _Alarmer_Signal
   barrier
   entry _Alarmer_Wait

end Alarmer


thread Sampler
--
-- This thread runs at 100 Hz in two modes, slow and fast.
--
```

```
-- In the fast mode, it samples 10 data points and sends
-- them to Averager.
--
-- In the slow mode, it samples one datum and sends it to
-- Averager. In addition, if this datum is above 290,
-- it signals an alarm.
--
   type cyclic
   root _Sampler

end Sampler


thread _Monitor
--
-- Monitors the average sampled value at 1 Hz.
-- If the average is below 50, signals an alarm.
--
   type cyclic

end _Monitor


thread _Reporter
--
-- Sporadic thread to report alarms.
--
   type sporadic

end _Reporter

end Monitor_Example
```

Example C code

To illustrate the relationship between the TPOF and the actual code, here is a possible C implementation of the *Sampler* thread, in the form of a subprogram that represents one cycle of the thread:

```
void Sampler (void)
/* This thread runs at 100 Hz in two modes, slow and fast.
 *
 * In the fast mode, it samples 10 data points and sends
 * them to Averager.
 *
 * In the slow mode, it samples one datum and sends it to
 * Averager. In addition, if this datum is above 290,
 * it signals an alarm.
 *
 */
{
   int datum, i;
   if (Mode == Slow) {
      datum = New_Sample();
      Averager_Add (datum);
      if (datum > 290) {
         Alarmer_Signal (datum);
      }
   } else {
```

```
      for (i = 0; i < 10; i++) {
         Averager_Add (New_Sample());
      }
   }
}
```

2.6 The Execution Skeleton

The HRT schedulability analyser needs, for each thread, a description of the worst-case execution path and a list of all the protected entries called (in any execution path). This is called the *execution skeleton* of the thread. The execution skeleton abstracts the overall worst-case execution path of one activation of the thread as a sequence of computations separated by protected entry calls. The sequence can contain loops with bounded repetition counts. Loops can be nested.

The execution skeleton cannot contain branching (alternative paths); it represents only the *overall* worst-case path. Other paths are represented only through the list of all protected entries called by the thread in any execution path.

The schedulability analyser also needs the execution skeleton of each protected entry, in the same form as for threads.

The purpose of the Bound-T HRT mode is to generate the execution skeletons of all threads and protected entries listed in the TPOF by analysing the corresponding subprograms. The skeletons are collected in the Execution Skeleton File (ESF) which is the main output of Bound-T in the HRT mode. Section 3.4 shows the ESF for the *Monitor_Example*.

3 USING HRT MODE

3.1 Inputs and Outputs

As in basic mode, Bound-T in HRT mode needs the compiled and linked form of the target program and may also need a file of user assertions to guide the analysis.

HRT mode needs, in addition, an input file that lists the threads and protected objects in the HRT structure of the target program. This is the TPOF (Threads and Protected Objects File). Chapter 2 showed an example TPOF. Chapter 4 defines the full TPOF syntax.

In HRT mode, Bound-T tries to bound the execution time of all threads and protected-object operations named in the TPOF. The results are displayed in the usual basic output format [1]. In addition, the results are combined with the TPOF into an Execution Skeleton File (ESF) which lists the threads and protected objects and provides a summary or skeleton of the worst-case execution path of each thread and protected operation.

The HRT schedulability analyser (not provided with Bound-T) uses the ESF to decide whether the target program is schedulable.

The following figure illustrates the inputs and outputs for Bound-T in HRT mode.

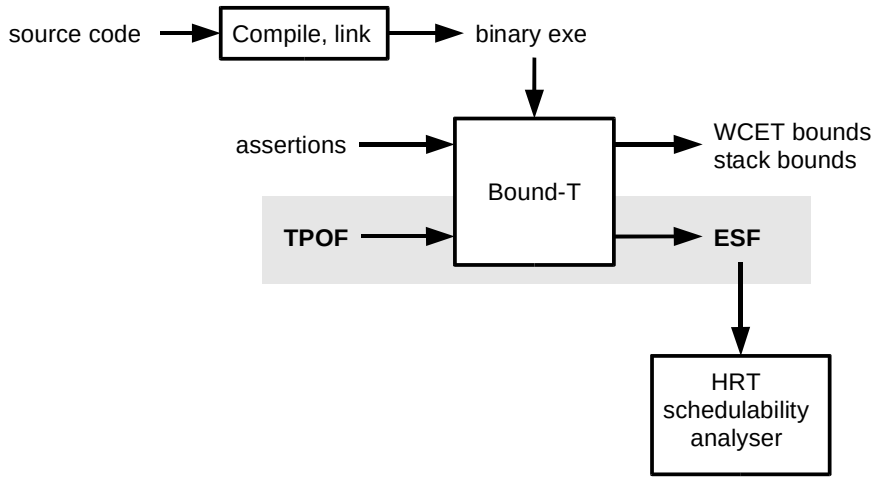

Figure 2: Inputs and Outputs for Bound-T in HRT Mode

An example ESF is shown at the end of this chapter. It corresponds to the example TPOF in Chapter 2. Chapter 5 defines the full ESF syntax.

3.2 Command Line

For the HRT mode of operation, the Bound-T command has the form:

```
boundt -hrt <more options> <target exe file> <TPOF name>
```

The option *-hrt* does not have to be first in the option list, but we recommend it for clarity because this is the part that separates the two modes of operation, basic mode and HRT mode.

<more options>

The other options are described in the Bound-T User Manual [1]. There are no special options for HRT mode, other than the option *-hrt* itself, and all basic-mode options can be used in HRT mode as well.

<target exe file>

The first argument after the options is the name of the file that contains the target program in linked, executable form, as in the basic mode.

<TPOF file>

The single remaining argument is the name of the text file that lists the threads and protected objects in the target program. These entities define the "root" subprograms for Bound-T.

3.3 HRT Mode Operations

When the *-hrt* option and the *<TPOF name>* argument are given, Bound-T works as follows.

Reading the TPOF

First, the definitions in the TPOF are checked and stored internally before any action is taken on them.

Analysing the WCET

Bound-T performs a normal WCET analysis of the subprograms that correspond to threads and protected entries, and their callees, exactly as if these subprograms were listed as root subprograms in the basic mode. All the basic and optional output from the analysis is produced just as in basic mode. These outputs are described in the Bound-T User Manual [1].

Generating the ESF

When the WCET analysis is complete, Bound-T generates an Execution Skeleton File (ESF) that contains the TPOF information extended with execution skeletons that describe the worst-case execution path of each thread and protected entry as a sequence of computations (with WCET bounds given) and protected calls, enclosed in a hierarchical looping structure.

File name conventions

The TPOF file-name is expected to end with the suffix ".tpo", and the ESF file-name is then constructed by replacing this suffix with ".esf". Otherwise, the ESF file-name is just the TPOF file-name extended with ".esf".

3.4 Example ESF

For the *Monitor_Example* program, the ESF might turn out as follows. Note that Bound-T copies all comments lines from the TPOF into the corresponding places in the ESF. The same does not hold for blank lines, however, and the indentation may change.

```
program Monitor_Example
--   Monitor: Tiny HRT Example for Bound-T
--
-- This program has three threads and two protected objects,
-- one resource object and one synchro object.

  protected Averager
    --
    -- The Averager object accumulates data samples into a sum
    -- and counts. Then, the average value can be read out,
    -- which also resets the sum and count.
    --
    type resource
    entry _Averager_Add
      wcet 52 , 7 , 8
    entry _Averager_Get
      wcet 111 , 11 , 14
  end Averager

  protected Alarmer
    --
    -- The Alarmer object provides a one-place buffer for an
    -- alarm value and Wait and Signal operations to wait for
    -- and signal an alarm.
    type synchro
    entry _Alarmer_Signal
      wcet 27 , 4 , 4
    barrier wcet tbd, tbd, tbd
    entry _Alarmer_Wait
      wcet 27 , 4 , 2
  end Alarmer

  thread Sampler
    --
    -- This thread runs at 100 Hz in two modes, slow and fast.
    --
    -- In the fast mode, it samples 10 data points and sends
    -- them to Averager.
    --
    -- In the slow mode, it samples one datum and sends it to
    -- Averager. In addition, if this datum is above 290,
    -- it signals an alarm.
    --
    type cyclic
      wcet 12 , 2 , 1
      loop 10
        wcet 74 , 0 , 4
        call_po Averager _Averager_Add
        wcet 85 , 2 , 5
      end
      wcet 8 , 2 , 0
    po Alarmer _Alarmer_Signal
  end Sampler
```

```
  thread _Monitor
    --
    -- Monitors the average sampled value at 1 Hz.
    -- If the average is below 50, signals an alarm.
    --
    type cyclic
      wcet 15 , 0 , 2
      call_po Averager _Averager_Get
      wcet 32 , 3 , 4
      call_po Alarmer _Alarmer_Signal
      wcet 38 , 5 , 4
  end _Monitor

  thread _Reporter
    --
    -- Sporadic thread to report alarms.
    --
    type sporadic
      wcet 6 , 0 , 2
      call_po Alarmer _Alarmer_Wait
      wcet 283 , 3 , 5
  end _Reporter
end Monitor_Example
```

4 SYNTAX OF TPO FILE

This chapter defines the formal syntax of the TPO file (TPOF).

Syntax notation

A conventional context-free syntax notation is used, with nonterminal symbols in Plain Style and Capitalised; literal keywords in **bold** style; and user-defined identifiers in *italic* style. Alternatives are separated by '|'. Repetition of a symbol for one or more times is denoted by a postfixed '+', and for zero or more times by a postfixed '*'. The symbol *null* stands for the empty string. The symbol *integer* stands for a string of digits 0 .. 9 representing an integer number in decimal form. Underscores '_' are also allowed in an *integer* but have no effect on the value of the number.

Right-hand-sides are often divided into several lines for clarity, but such lay-out is not syntactically significant.

Syntax productions

The start symbol is TPOF, representing the whole file.

TPOF →
    program *prog_name*
        Definition +
    end *prog_name*

Definition → Thread_Def | Resource_Def | Synchro_Def

Thread_Def →
    thread *thread_name*
        type Thread_Type
        Optional_Root_Name
    end *thread_name*

Optional_Root_Name → *null* | **root** *root_subprogram_name*

Thread_Type → **cyclic** | **sporadic** | **interrupt_sporadic**

The type of the thread is not necessarily significant for Bound-T, but Bound-T will copy it to the generated ESF for completeness. If a *root_subprogram_name* is not given, the *thread_name* is used as the name of the root subprogram.

Resource_Def →
    protected PO_name
        type resource
        PO_Entry +
    end PO_name

Synchro_Def →
    protected *PO_name*
        type synchro
        PO_Entry
        Barriered_PO_Entry
    end *PO_name*

The first PO_Entry is the "activate" entry. The Barriered_PO_Entry is the "wait for activation" entry.

PO_Entry → **entry** *entry_name*

Barriered_PO_Entry → **barrier** PO_Entry

Lexical rules

On the lexical level, the TPOF is assumed to be a normal text file (ASCII or Latin-1 coding, system-defined line separators). Keywords are case-insensitive and are not reserved (thus a user-supplied identifier can match a keyword). User-supplied identifiers are case-sensitive and can contain any graphic character except space. White space between symbols can be either blanks, tabs or line separators.

The TPOF can contain comments in the Ada style, starting with the double hyphen "--" and extending to the end of the line. TPOF comments are automatically copied to the generated Execution Skeleton File.

5 SYNTAX OF EXECUTION SKELETON FILE

This chapter defines the formal syntax of the Execution Skeleton File (ESF) that Bound-T generates. It contains the same information as the user-supplied TPOF, extended with execution statements and WCET bounds.

The resulting ESF language is intended to be the same as in the HRT reference documents although the grammar has a slightly different form.

Syntax notation

The same syntax formalism is used as for the TPO file in Chapter 4.

To distinguish similar nonterminals in the TPOF and ESF, the prefix "ES_" is added to ESF nonterminals.

Syntax productions

The start symbol is ESF, representing the whole Execution Skeleton File.

ESF → **program** *prog_name*
    ES_Definition +
end *prog-name*

ES_Definition → ES_Thread_Def | ES_Resource_Def | ES_Synchro_Def

ES_Thread_Def →
thread *thread_name*
    type Thread_Type
    Execution_Skeleton
end *thread_name*

Execution_Skeleton →
Execution_Statement_List
Optional_Protected_Object_Calls

Execution_Statement_List → Execution_Statement +

Thus, the execution skeleton of a thread (or protected entry, see below) consists of two parts: firstly, a list of *execution statements* describing the worst-case execution, including loops and protected entry calls, and secondly, a list of *other protected entry calls* – if any – that may occur in other executions.

Execution_Statement →
    wcet Time
| **call_po** *PO_name entry_name*
| Loop_Statement

Time → Processing_Cycles **,** Memory_Reads **,** Memory_Writes

Processing_Cycles → *integer*

Memory_Reads → *integer*

Memory_Writes → *integer*

The Time measure is divided into three components: number of processing cycles (CPU cycles), number of memory reads, and number of memory writes. The reads and writes are separated to let the HRT schedulability analyser account for memory wait states. Note that the number of processing cycles (the first component) already includes the effect of the wait states that were specified for the Bound-T analysis (by target-specific options or assertions).

The two commas in the production for Time are terminal tokens that are generated in the output file.

Loop_Statement →
    loop Loop_Count
        Execution_Statement +
    end

Loop_Count → *integer*

The Loop_Statements show the loops that Bound-T has found in the control-flow graph of the subprogram. The Loop_Count is an upper bound on the number of executions of the loop body, and may result from Bound-T's automatic loop-bound analysis or from an assertion.

The ESF syntax describes a loop that tests for termination before or after the loop body, but not in the middle of the loop body. If the control-flow graph contains a loop with a test in the middle, Bound-T transforms the loop into the ESF form by duplicating the part of the loop body that precedes the termination test and exit.

Optional_Protected_Object_Calls → *null* | **po** PO_Entry_Ref +

PO_Entry_Ref → *PO_name entry_name*

ES_Resource_Def →
    protected *PO_name*
        type resource
        ES_PO_Entry +
    end *PO_name*

ES_Synchro_Def →
    protected *PO_name*
        type synchro
        ES_PO_Entry
        ES_Barriered_PO_Entry
    end *PO_name*

ES_PO_Entry →
    entry *entry_name*
        Execution_Skeleton

ES_Barriered_PO_Entry →
    barrier wcet Barrier_Time
    ES_PO_Entry

Barrier_Time → **tbd , tbd , tbd**

The WCET time for barrier evaluation (Barrier_Time) cannot be computed automatically by Bound-T. Although the barrier is just a boolean expression, and Bound-T can surely compute the WCET of a boolean expression, the expression is probably embedded in compiler- and kernel-specific code that would need specific recognition and parsing. Note that the three **tbd** tokens occur as such in the generated ESF and must be edited by the user before schedulability analysis.

Lexical syntax

Bound-T generates the ESF as a normal text file, like the TPOF.

6 TROUBLESHOOTING

This section explains how to understand and correct problems that may arise when using Bound-T in HRT mode, by listing those warning and error messages that are specific to HRT mode and explaining what they mean and what to do in each case.

All the warning and error messages that can arise in basic mode can also arise in HRT mode. Please refer to the Bound-T User Manual [1]. Here we discuss only the messages related to the HRT mode.

If you cannot find a particular message here or in the User Manual, please refer to the Application Notes for your target system and host platform; additional, target-specific messages may be listed there.

6.1 Bound-T Warning Messages for HRT Mode

The following table lists all Bound-T's HRT-related warning messages in alphabetical order using the same format as in the Bound-T User Manual [1].

Table 1: HRT-Related Warning Messages

Warning Message		***Meaning and Remedy***
TPOF line *L* col *C* : *message :* Token : *token*	*Reasons*	A problem, described in the *message*, has been identified in the TPO file on or after line number *L* and column number *C*, while processing the current lexical *token*.
	Action	Correct the TPO file.

6.2 Bound-T Error Messages for HRT Mode

The following table lists all Bound-T's HRT-related error messages in alphabetical order using the same format as in the Bound-T User Manual [1].

Table 2: HRT-Related Error Messages

Error Message		***Meaning and Remedy***
Barriered entry subprogram *name* is not found	*Problem*	In the TPO file, the *name* of the barriered entry (waiting entry) does not match any subprogram in the target program.
	Reasons	Error in the TPO file, or some name mangling by the compiler and linker.
	Solution	Correct the TPO file to use the same subprogram name as in the target program executable.
HRT: *number* contradictions in HRT structure.	*Problem*	The stated *number* of inconsistencies were found in the user-supplied TPOF or between the TPOF and the target program. The errors are also shown by earlier, specific error messages. The analysis is therefore aborted.

Error Message		*Meaning and Remedy*
	Reasons	The TPOF or the program is incorrect or violates HRT design rules.
	Solution	Correct the TPOF and/or the program as indicated by the earlier, specific error messages.
No TPOF name given.	*Problem*	The *-hrt* option was given to Bound-T at start-up but no TPOF name was given.
	Reasons	Perhaps the user has misunderstood the command syntax.
	Solution	Restart with correct arguments.
Option conflict: HRT analysis requires time bounds.	*Problem*	HRT analysis is requested (with the *-hrt* option) but execution-time analysis is disabled (with the *-no_time* option). This is contradictory.
	Reasons	Mistake on the command line.
	Solution	Correct the command-line options.
Protected object *P* calls a thread *T*	*Problem*	In the HRT model, it is impossible for a protected object to call a thread, but this program does this.
	Reasons	Probably mistaken names in the TPOF.
	Solution	Correct the TPOF.
Thread *T1* calls another thread *T2*	*Problem*	In the HRT model, it is impossible for one thread to directly call another, but this program does this.
	Reasons	Probably mistaken names in the TPOF.
	Solution	Correct the TPOF.
TPO file *name* could not be opened for reading.	*Problem*	In HRT analysis, the TPO file could not be opened for reading under the file-*name* the user gave.
	Reasons	Wrong file-name or read-protected file.
	Solution	Correct the file-name or modify the protection of the file.
TPO file *name* was not found.	*Problem*	In HRT analysis, the TPO file was not found under the file-*name* the user gave.
	Reasons	Wrong file-name or protected directories on the path to the file.
	Solution	Correct the file-name or modify the protection of the directories.
TPO file contained subprograms that were not found.	*Problem*	In HRT analysis, one or more of the threads (or their roots) or protected object entries did not have the same name as some subprogram in the target program. Each problem has been separately reported earlier.
	Reasons	Probably mistakes in the TPO file, or name mangling by the compiler.
	Solution	Correct the TPO file to use the subprogram names as they appear in the target program executable.
TPO file contained total of *number* syntax errors.	*Problem*	The TPO file has a *number* of syntax errors, which have been reported separately earlier.
	Reasons	Incorrect TPO file, or name mangling by the compiler leading to mismatch between source-level subprogram names and executable-level names.
	Solution	Correct the TPO file as indicated by the specific errors reported earlier.

Error Message		*Meaning and Remedy*
TPOF line *L* col *C* : *message* : Token: *token*	*Problem*	An error, described in the *message*, was discovered during the parsing of the TPO file, on or after line number *L* and column number *C*, while processing the current lexical *token*.
	Reasons	The specific reason is indicated in the *message* given. Typically a syntax error in the TPO file.
	Solution	Edit the TPOF to correct the problem indicated in the message.
TPOF name does not end with '.tpo'	*Problem*	In HRT analysis, the user-supplied name of the TPO file, which is the last command argument <TPOF name>, does not end with the suffix ".tpo".
	Solution	Correct the file name, or accept the resulting perhaps non-standard ESF name (TPOF name plus ".esf").
Unable to create execution skeleton file *name*.	*Problem*	In HRT analysis, Bound-T was unable to create the Execution Skeleton File, because the *name* was invalid, or the file was not writable, or the file had invalid status.
	Reasons	Write-protected directory or file, internal fault, or operating system problem.
	Solution	Check the protection on directories and the ESF and correct (allow writing).
Unique edge expected, but second edge found (*new* after *first*)	*Problem*	The worst-case path contains alternative branches (other than loop exits) and cannot be represented in an Execution Skeleton. The two numbers, *first* and *new*, are the indices of two flow-graph edges that have the same source node.
	Reasons	Some assertions or analysis force Bound-T to include alternative paths through some loop body (different paths on different iterations). For example, the reason may be an assertion that limits the number of executions of the worst-case path in the loop body to less than the number of repetitions of the loop, forcing Bound-T to use another path for the rest of the iterations.
	Solution	Change the assertions to make the worst-case path use the same path in each loop body for all iterations of the loop.

7 GLOSSARY

This glossary contains only HRT-related terms and abbreviations. Please refer also to the glossary in the Bound-T User Manual [1].

ESF — Execution Skeleton File. The text file generated by HRT-mode analysis of an HRT target program and containing the information from the TPOF supplemented with execution skeletons containing WCET values. See chapter 5.

HRT — Hard Real Time; a principle for real-time program architecture, and a theory and tool-set for analysing such programs. An HRT program consists of threads and protected objects. See chapter 2.

Protected object — A component of an HRT program that is a passive entity and acts as a communication and synchronisation point for threads. See chapter 2.

Rate-Monotonic Analysis — A way to analyse the schedulability of a multi-threaded program where the threads are periodic and scheduled by priority with pre-emption. Rate-Monotonic Analysis (RMA) assigns priorities to threads monotonically in order of thread period so that short-period, high-rate threads have higher priorities than long-period, low-rate threads. With such a priority assignment the WCETs of the threads can be plugged into mathematical formulae that show if the thread set is schedulable (each thread can execute to completion without overrunning its period).

RMA — See *Rate-Monotonic Analysis.*

Scheduling — The allocation of processor resources (execution time) to the several threads in a concurrent program. Specifically, the selection of which thread shall be running at every moment.

Task — See *thread.*

Thread — An active component of a program, executing program statements sequentially. Some programs have a single thread of exection, but many real-time programs are multi-threaded, with several threads executing concurrently. The number of threads that can be (truly) executed in parallel depends on the number of processors in the target system.

For Bound-T, the usual assumption is that there is one processor, which is shared among the threads via thread scheduling. See chapter 2.

TPO file — Threads and Protected Objects File. See *TPOF*.

TPOF — Threads and Protected Objects File. The user-supplied text file that lists and describes the structure of an HRT program, for HRT-mode analysis by Bound-T. See chapter 4.