**TIMES-Pro: A Tool for Modeling, Analysis, Simulation, and Implementation of**

**Real-Time Systems**

[Department of Information Technology](http://www.it.uu.se/)

Uppsala University

2016

Table of Contents

[**1. Introduction 3**](#_Toc466294111)

[**2. A Review on DRT and SDRT 4**](#_Toc466294112)

[**3. Installation 5**](#_Toc466294113)

[**4. Designing a Real-Time Task Set 6**](#_Toc466294114)

[**5. Specifying the Architecture 9**](#_Toc466294115)

[**6. Simulation 10**](#_Toc466294116)

[**7. Code Generation 11**](#_Toc466294117)

[**Appendix A. Software Architecture 17**](#_Toc466294118)

# Introduction

TIMES-Pro is a tool for **m**odeling, **a**nalysis, **s**imulation, and **i**mplementation of real-time tasks. It provides a means to specify real-time task graphs through a graphical user interface (GUI). The tool provides a simulation facility to show the user the possible timing behavior of the tasks when scheduled by a fixed- or dynamic-priority scheduling policy.

Currently, the tool supports one of the most expressive workload models of real-time systems, namely SDRT. In the SDRT model, a real-time task is specified by a graph which can have synchronizations with other task graphs.

# A Review on DRT and SDRT

A review on the DRT and SDRT task models and the respective analyses goes here.

Nomenclature:

|  |  |
| --- | --- |
| **Notation** | **Definition** |
|  | The set of vertices (job types) of a task |
|  | The set of edges of a task |
|  | The WCET of job type |
|  | The relative deadline of job type |
|  | The label of an edge , representing the inter-release time between job types and v . |
|  | The synchronization action of an edge |

# Syntactical Rules

A given set of digraphs is considered as a valid SDRT task set if the following conditions are satisfied[[1]](#footnote-2).

1. , (), and are integers for all and .
2. .
3. .
4. .
5. Task names are unique.
6. The name of a job is unique with respect to the other jobs in the same task.
7. Task priorities should be integer values.
8. Tasks are subject to constrained deadlines, that is, .
9. All the names should follow the Naming Rule (defined below).
10. For each sending action, there exists at least one receiving action in some another task.
11. An action cannot appear in a task more than once.

**Naming Rule:**

<digit> ::= “0” | “1” | … | “9”

<char> ::= “a” | … | “z” | “A” | … | “Z” | “\_”

<string> ::= <char> | <digit> | <char><string> | <digit><string>

<name> := <char> | <char><string>

## Rules Specific to Analysis

The following rules are specific to analysis. Meanwhile, for other purposes (like code generation), maybe they do not need to be respected.

* For any , there exists at most one edge from to . In other words, does not contain redundant elements.
* .

Rules required for code generation:

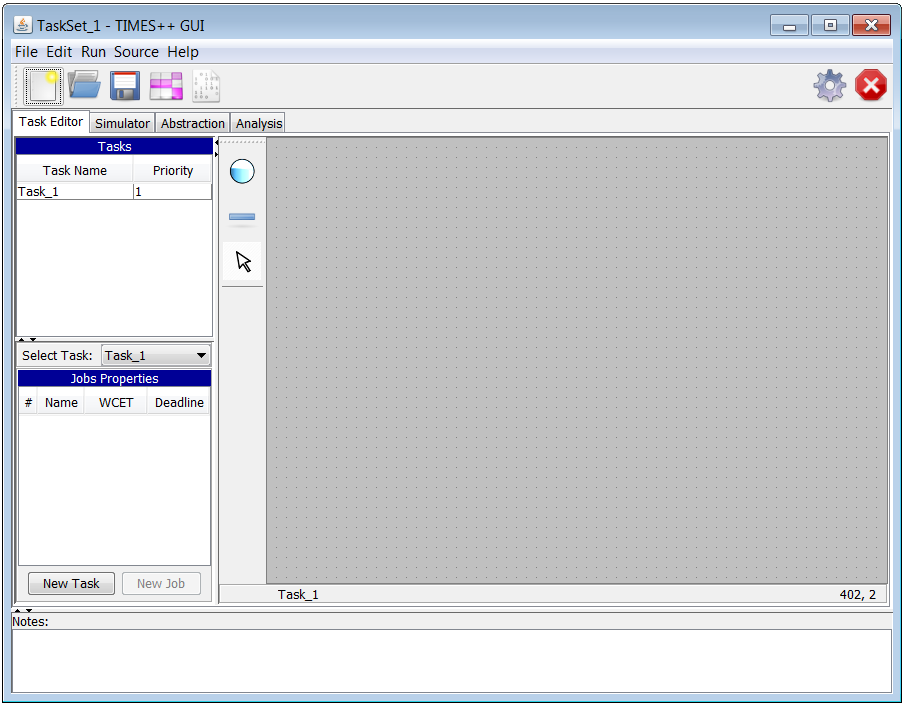
# Installation

The system prerequisites and required packages (like JRE) are explained here. In addition, the installation instructions are presented.

# Designing a Real-Time Task Set

Here, we will explain that how the model of a sample DRT task set can be specified in the tool. We will show some snapshots of the corresponding steps.

Some examples are seen below.



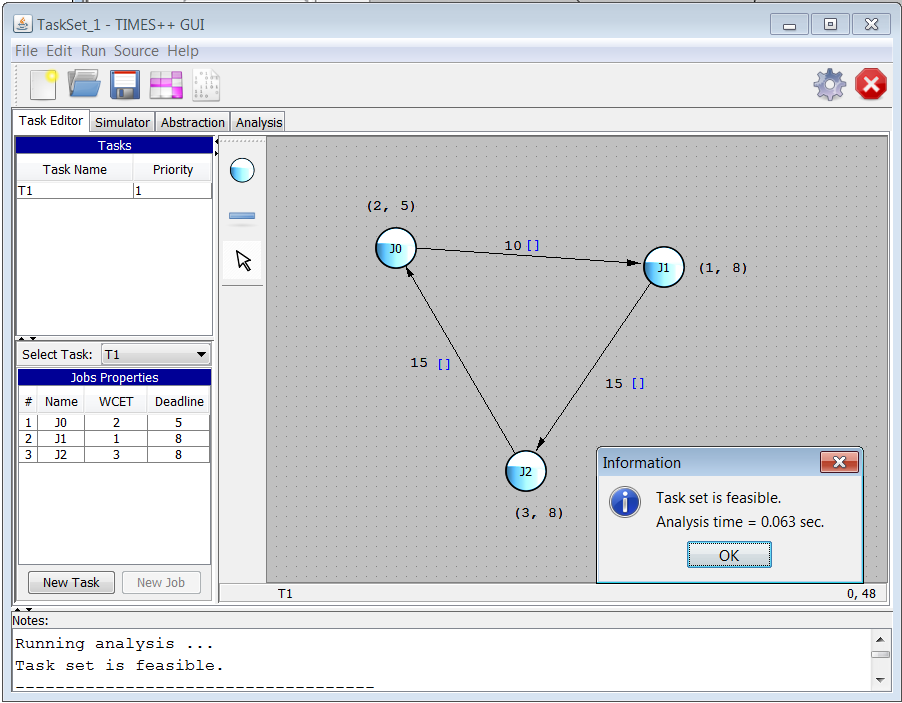
**1**

**2**

**3**

**4**

1. Create new task set
2. Open an already saved task set (xml format)
3. Random task generation
4. Create new task in the existing task set



**1**

**2**

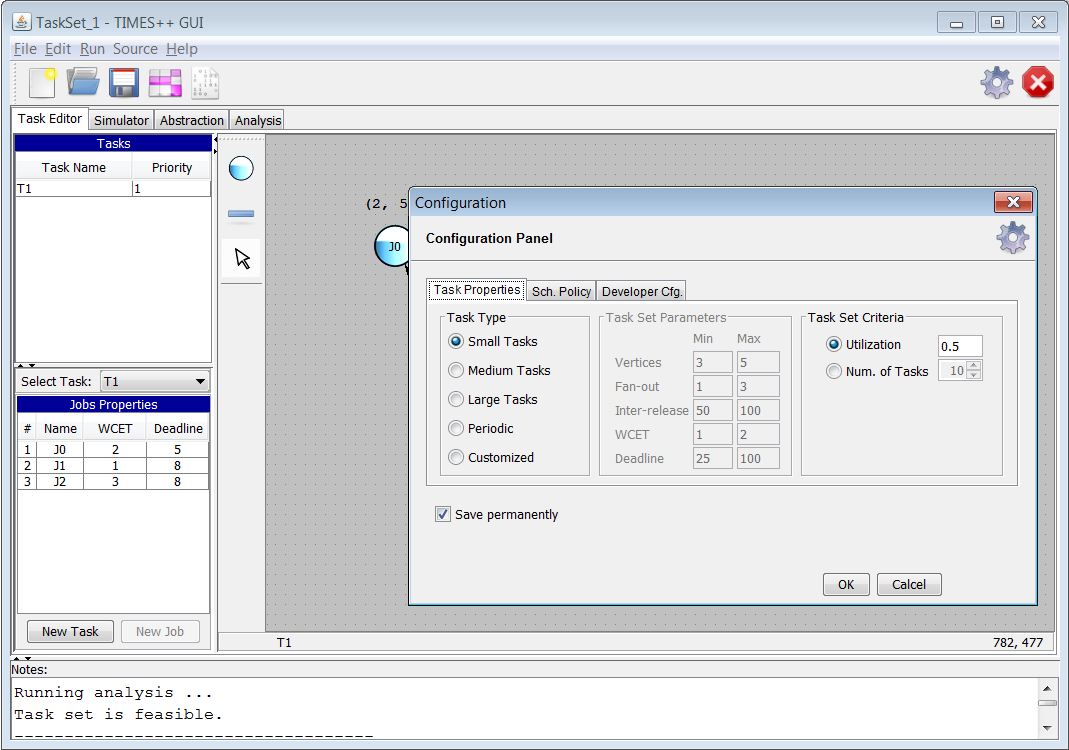
**3**

**5**

**4**

**6**

1. New node (job type).
2. New edge.
3. Default pointer.
4. Run schedulability test.
5. The analysis result is printed here.
6. The analysis time, as well as the result is shown here.



**1**

**2**

**3**

1. Open the configuration panel.
2. Parameters of random task generation
3. Scheduling policy (used for the schedulability analysis)

**Random task set generation**

Through this feature, the user can specify a range of task parameters. The tool then will generate a random task set in accordance with those parameters.

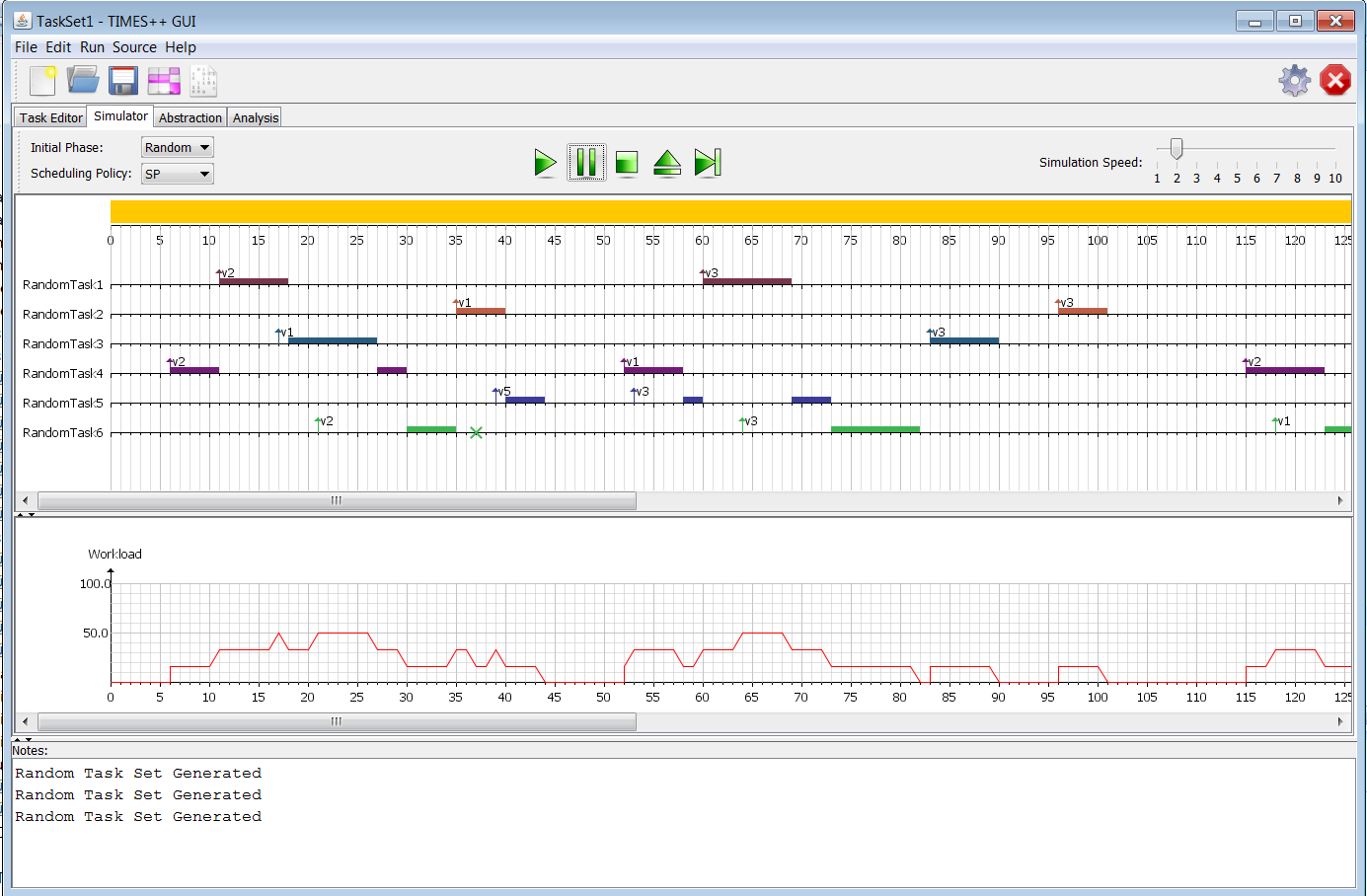
The user can select the size of the task set, its utilization, or both. If both are selected, the system first generates a task set with the specified number of tasks; then the parameters (minimum inter-release times) are scaled such that the desired utilization is achieved.

# Specifying the Architecture

# Simulation

The respective explanations for the simulation tab go here.

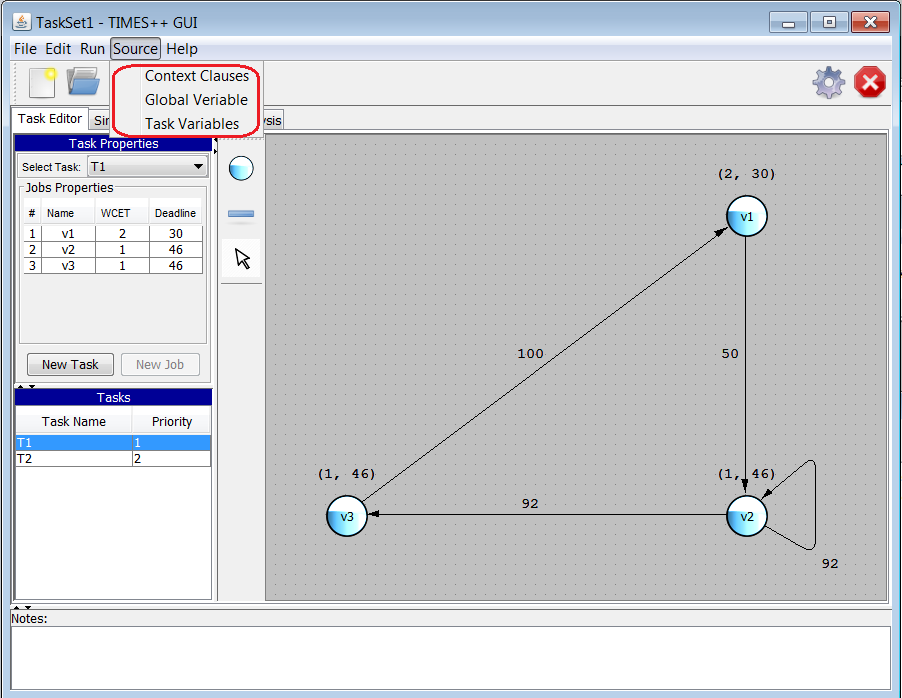
A sample trace of the simulation of 6 tasks with fixed-priority policy is seen in the figure below.

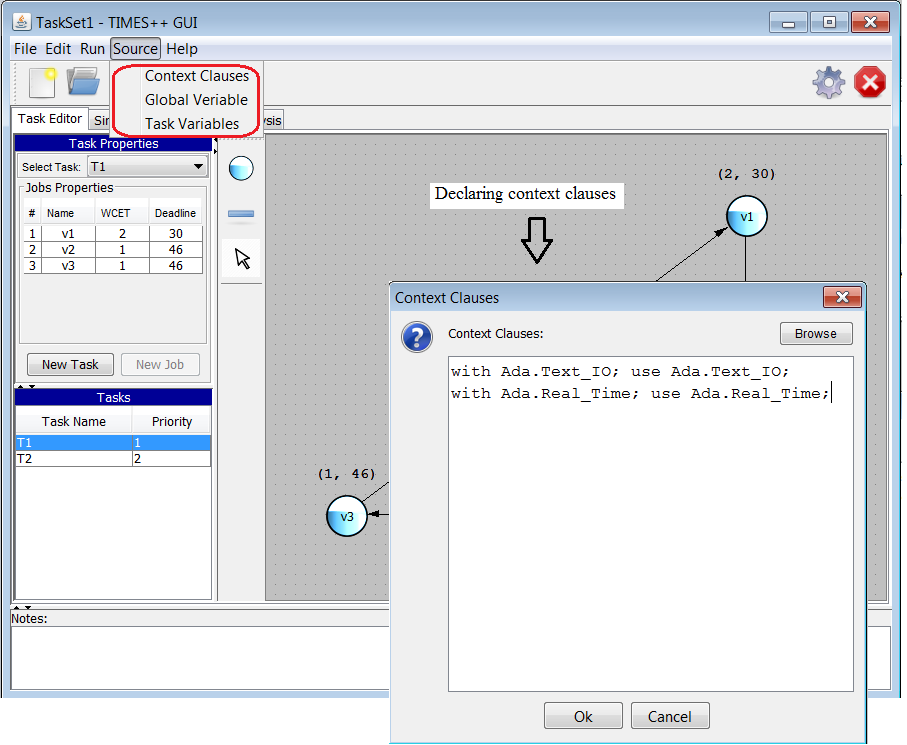


# Code Generation

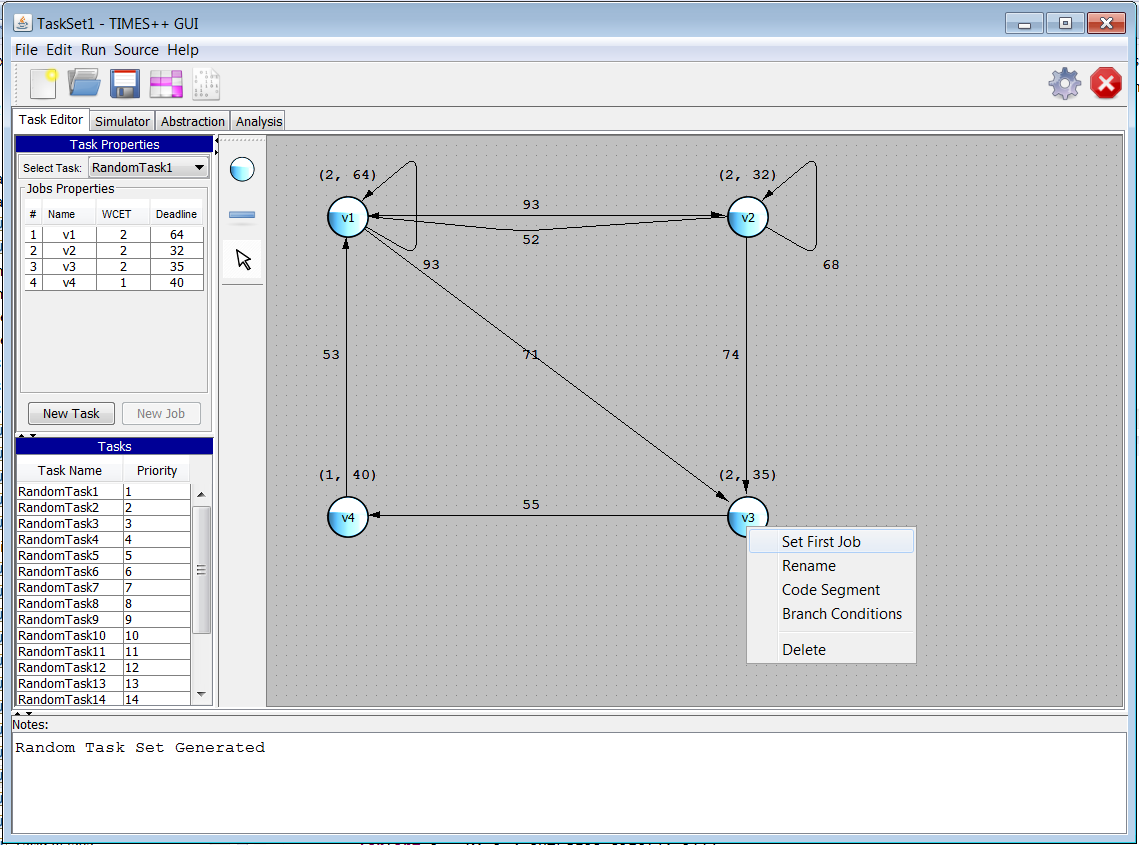
The “Code Generation” feature of TIMES-Pro provides the facility of producing Ada code from a modeled task set. This section provides a set of snapshots from the tool to guide the user to use this facility. Code generation can include the following steps, which are depicted in the subsequent figures:

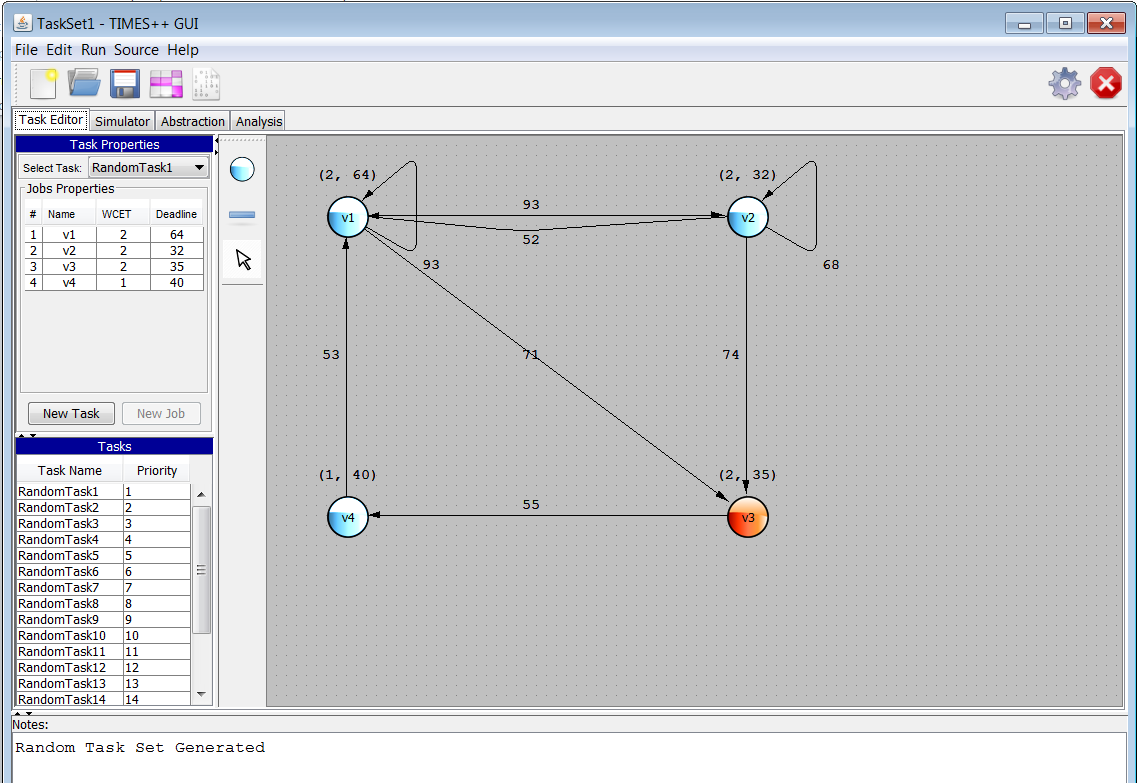
1. Declaration of context clause, global variable, and local variables;
2. Setting the starting job of each task;
3. Assign a code segment to each job;
4. Assign a logical condition to each branch;
5. Code generation.
6. **Declaration of context clause, global variable, and local variable:** Using this feature, a user can specify (declare) context clauses (for instance “with clauses” to use a certain Ada library), a set of global (system-wide) variables, as well as task specific (local) variables, as shown in the following three pictures.



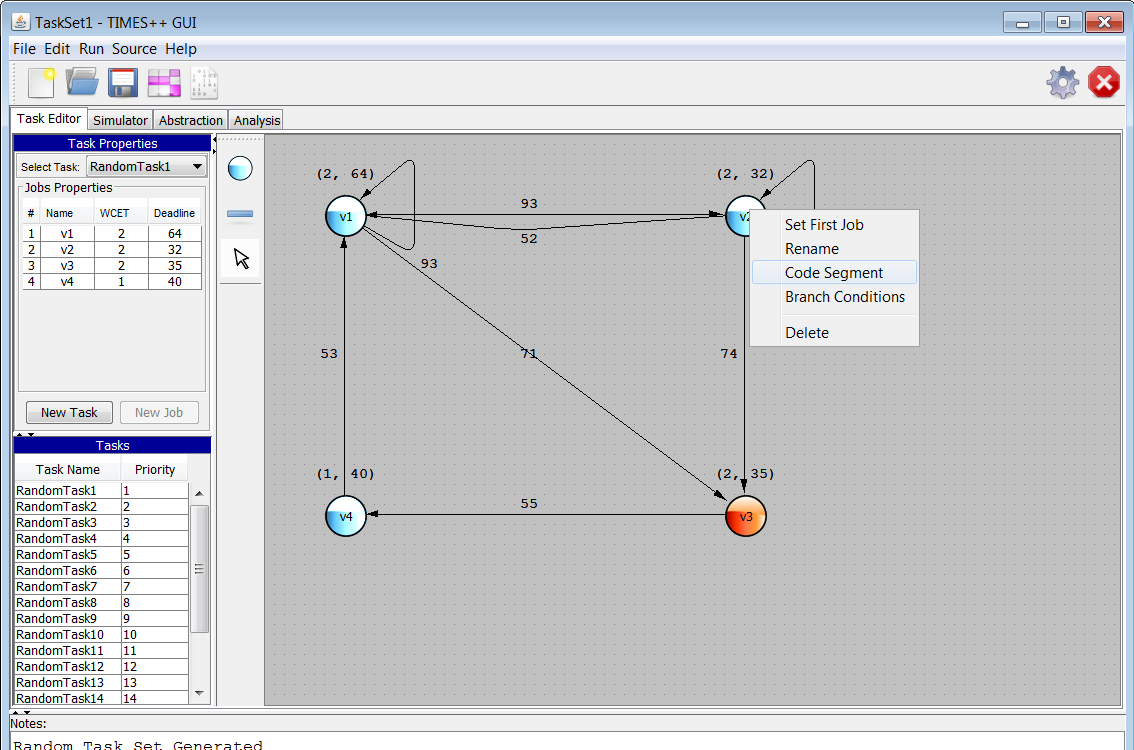


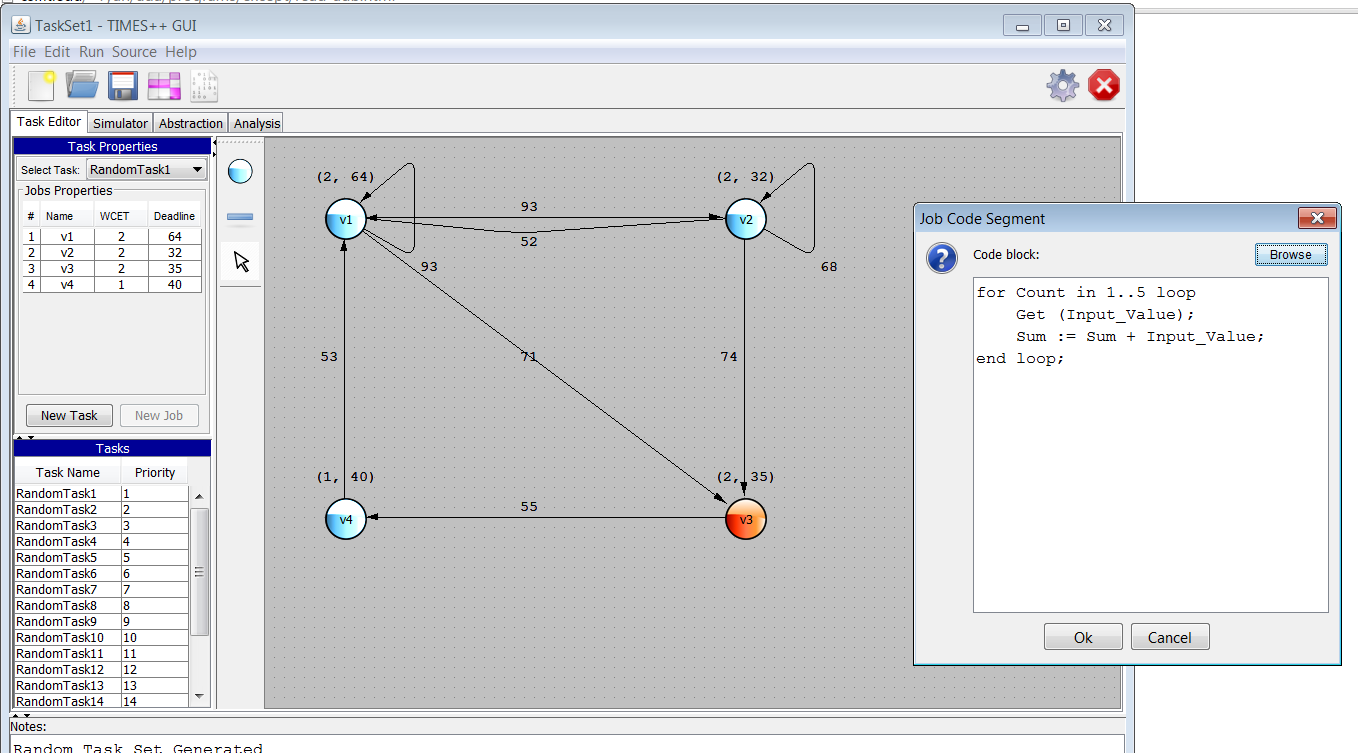
1. **Setting the first job of a task:** Used to make a vertex as the starting job of the task. The starting job (vertex) is distinguished by a different color.



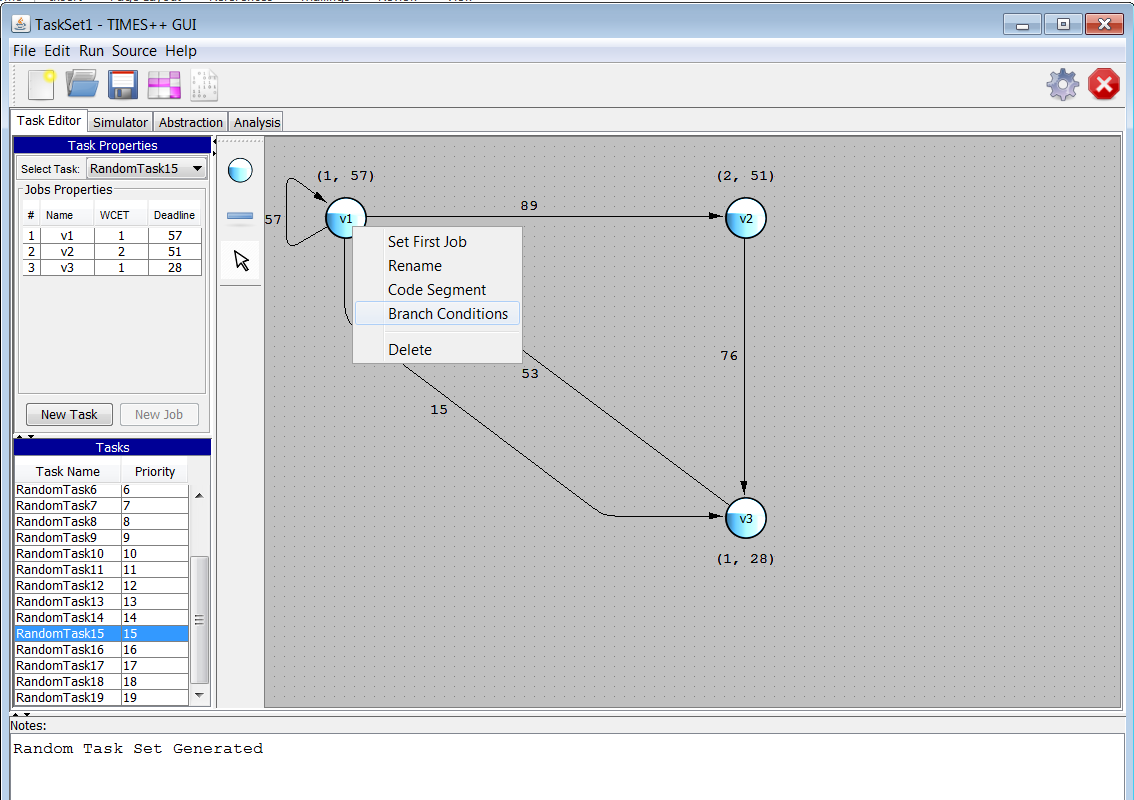


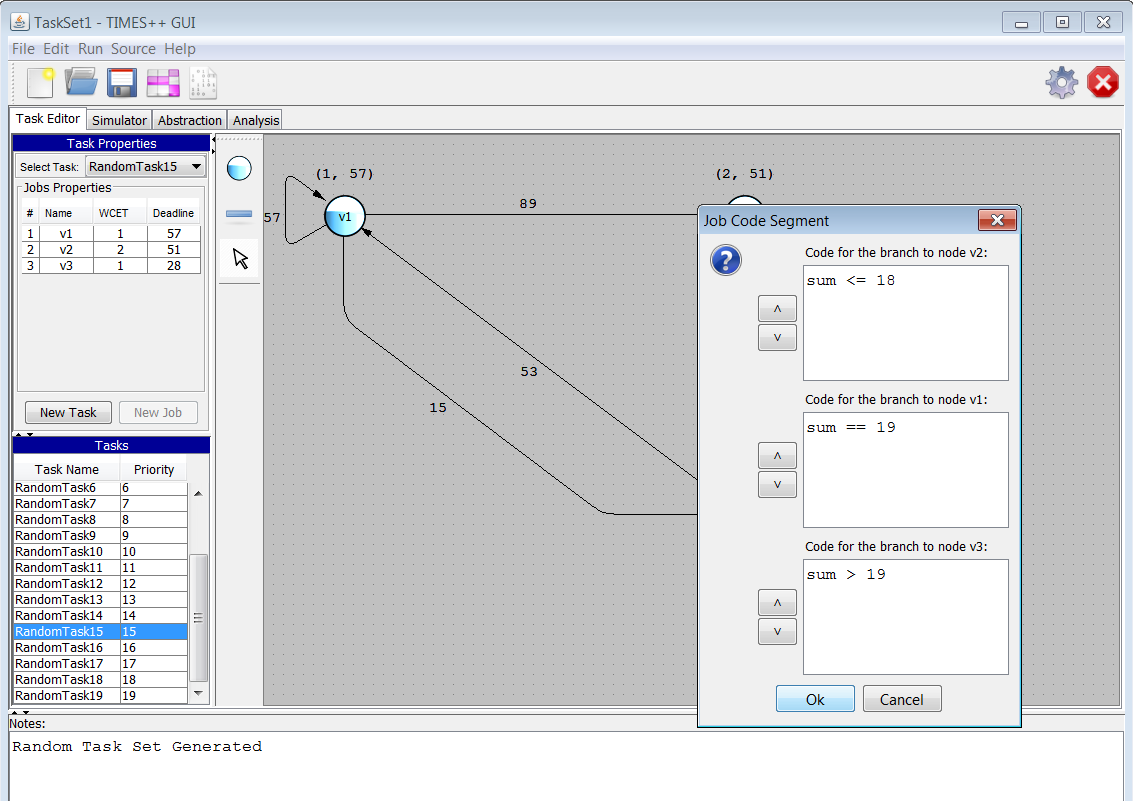
1. **Assign a code segment to a job:** The user can assign a code segment to each jobtype (vertex).



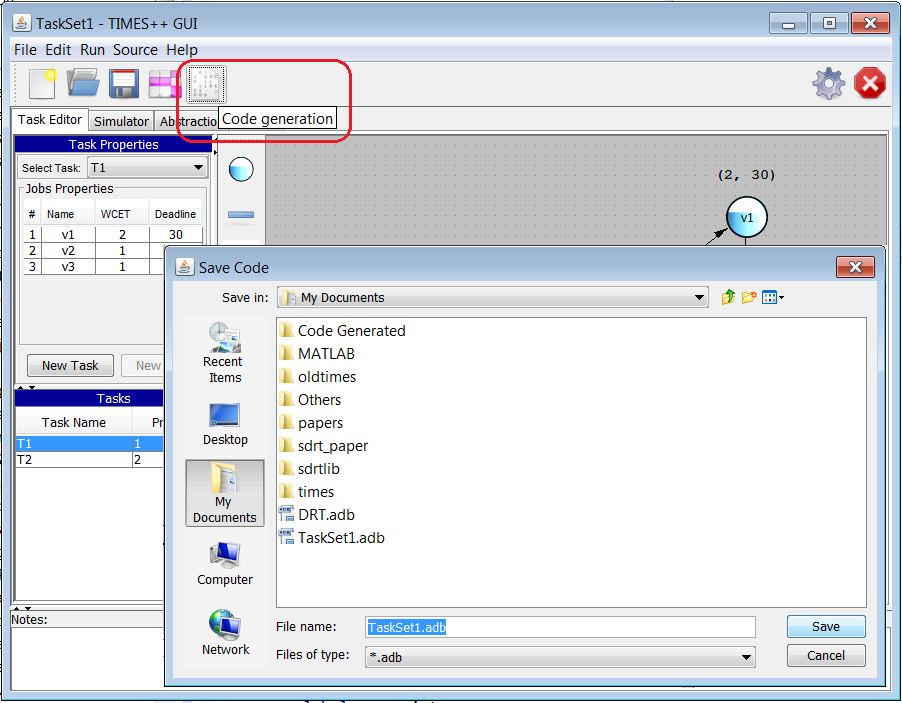


1. **Branch conditions:** In a DRT task graph, there may be more than one outgoing edge from a specific vertex. This situation represents a branch in the program. The user needs to provide logical conditions to determine which branch should be taken at run time. The order in which the conditions are checked can also be determined by the user.

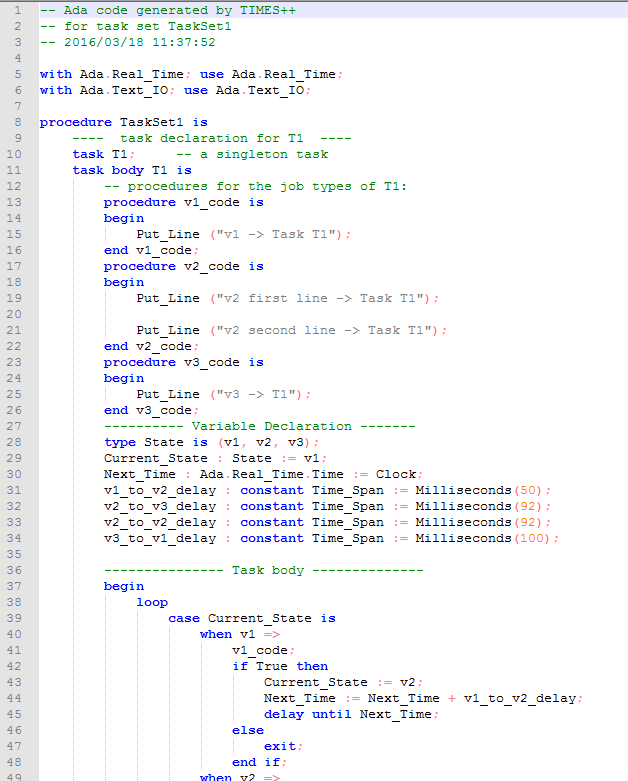




1. **Code generation:** According to the above mentioned inputs from the user, the tool generates an Ada source file which contains the code for the current task set.



The following image shows a part of the Ada code generated by the tool for a simple task set containing two DRT tasks.



# Appendix A. Software Architecture

This section briefly reviews the main components of the software and their communication mechanism.

Java front end

Python back end

**Task Editor**

**Simulator**

**Code generator**

XML file

package storage:

class TSLoader;

class TSSaver;

rndtask.py

rf\_dbf.py

partition.py

responsetime.py

class Manager;

Ada source files

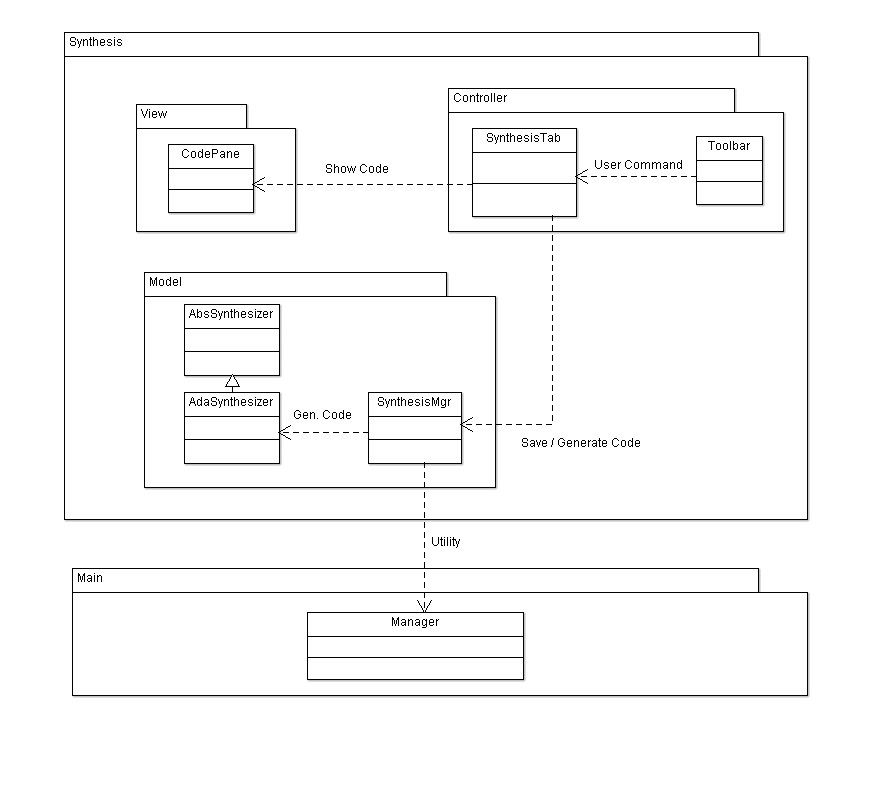
class AdaSynthesizer;

class Util;

**Design Pattern of the Code Synthesis Component:**

There is a separate tab for code generation. The classes involved in this tab are arranged in the way presented in the figure below.

In terms of the MVC (Model-View-Controller) design pattern, the package structure can be represented as follows:



A design rule in this architecture is that any changes related to the domain model (including: code generation, saving, …) should be done through “SynthesisMgr”, and not by the classes in the Controller layer.

**Important packages/components:**

* Dialogs.java: a class containing all dialog panels to show to the user.
* Package service: this contains classes to run external programs, i.e., python, gnat compiler, etc.

1. Items in blue are checked immediately whenever the user makes a modification. Items in black (as well as the syntax rules specific to analysis) are checked whenever explicitly requested by the user. Items in green are not implemented yet. [↑](#footnote-ref-2)