Image Cover Sheet

CLASSIFICATION	SYSTEM NUMBER 515029
UNCLASSIFIED	

TITLE

Air traffic control task - user guide and system overview

System Number:

Patron Number:

Requester:

Notes:

DSIS Use only:

Deliver to:

This page is left blank

This page is left blank

DCIEM No. CR 2000-121

AIR TRAFFIC CONTROL TASK – USER GUIDE AND SYSTEM OVERVIEW

by

Marc S. Grushcow

NTT Systems Inc.
86 Dunblaine Ave., Toronto, ON
Canada, M5M 2S1

PWGSC Contract No. W7711-8-7583/001/TOR

on behalf of
DEPARTMENT OF NATIONAL DEFENCE

As represented by
Defence and Civil Institute of Environmental Medicine
1133 Sheppard Avenue West
Toronto, Ontario, Canada
M3M 3B9

DCIEM Scientific Authority:
Name: Keith C. Hendy
Telephone no. (416) 635-2074

31 March 2000

Air Traffic Control Task

User Guide

and

System Overview

2000 03 31

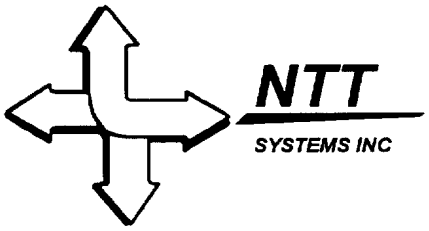
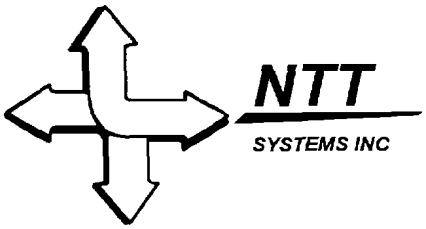
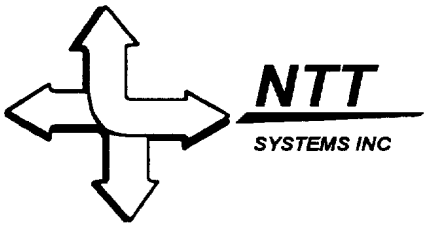


Table of Contents





1. Introduction

The Air Traffic Control Task was originally developed by Somapro Ltd., in C for the Mac. This document describes a re-implementation of the task in Java, a language that runs on a variety of computer architectures, including PC/Windows, the DND standard. The goal was to create a functionally equivalent version of the task while redeveloping certain hard to use features.

We are assuming that the reader is familiar with ATC. If that is not the case, please refer to the Somapro documentation. The purpose of this document is to...

1) Describe how to configure and run the task and interpret its output.
2) Install the task on a WinTel computer.
3) Provide an overview of the software architecture and references to more detailed information.

2. Installation Procedure

The application and required Java runtime files (WinTel) are provided in a single ZIP file. Create an empty directory anywhere on any local hard drive and unzip the files, making sure to use the option that will recreate the subdirectory structure from the ZIP.

3. User Guide

In the previous version of ATC, the Experimenter had to set up all task parameters by starting the task, entering setup mode and manually configuring all options through a series of dialogues. The configuration was then saved and restarted when it was run for a subject. This made it very difficult to set up and verify a suite of configurations that would implement the different conditions of an experiment.

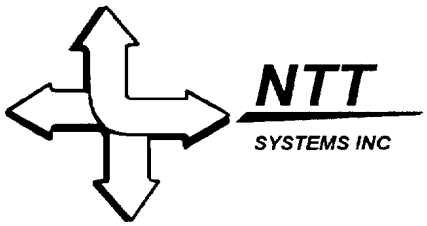
This version of ATC addresses the configuration problem through configuration files — one that defines the User Interface and the interaction between the subject and the system, and one that defines parameters that can be used to implement different experimental conditions. A third, optional, file can be used to specify aircraft characteristics explicitly as opposed to statistically.

This portion of the document explains how these are used, how to run experiments, and how to interpret ATC result files.

3.1 Required and Optional Files

3.1.1 The .GUI File

This file defines all of the components that make up the ATC Graphical User Interface. It is effectively the information that makes ATC look like ATC. It is possible to construct different GUI's that use the



same underlying ATC simulation. Experimenters might use this to investigate how different GUI's affect user performance. ATC uses much of the same software developed for TITAN, another task developed for DCIEM. Refer to http://www.ntt.ca/titan.html for examples of creating different GUI's that run over the same TITAN simulation.

NOTE: *As an Experimenter, you should regard the GUI file as part of the ATC software. You have to refer to it in .INI files, but under no circumstances should you attempt to modify it. You should be aware of the fact that the GUI* can *be changed if you need different GUI elements or interactions.*

3.1.2 The .INI File

The .INI file contains all of the information that defines how a particular ATC condition will behave. All experimental trials that use the same input files will behave the same way, including repeating the same randomization sequences. In particular that means that all generated aircraft information will be reproduced in each run.

The .INI file is a set of parameter name/value pairs. It can be created with any **ASCII** text editor. The following format rules apply:

1) Case is not significant.
2) Lines beginning with a "!" are considered to be comments.
3) A parameter and its value must be separated by one or more spaces or tabs and must be coded on a single line.
4) All values must be enclosed in double quotation marks (").
5) Lines in the .INI file may appear in any order.

The following is a complete list of annotated parameters along with their allowable values or syntax. Lines that would appear in the .INI file are in **`Courier bold`** font and annotations are Times Roman.

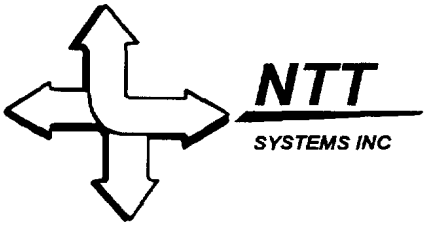
3.1.2.1 Required File References

`LeaderGuiScript "guiScript/Atc/LeaderSolo.gui"`

This parameter specifies the GUI configuration file. In this case, the file is located *relative* to the home directory.
VALID SET OF VALUES: Any valid GUI configuration file. Note that all file names are specified using the *forward slash,* regardless of operating system. Paths may be absolute or relative.

`Logfile "c:/temp/ATC.log"`

This parameter specifies the name of the result file. It *must not* exist when ATC is started. ATC will exit with an error message and refuse to overwrite. EXCEPTION: ATC will overwrite a file named `titan-demo.log`, to facilitate training and demonstration runs. In this example, the result file is located in a completely specified path, i.e., not relative.
VALID SET OF VALUES: Any syntactically valid file name. User forward slashes in all cases.



3.1.2.2 Result File Content

Header "XCXCXCXC"

The header is written as the first column of every line in the result file. This allows data to be easily combined for analysis. Typically the header is used to encode subject and experiment condition information.

VALID SET OF VALUES: any string

LogActionInterface "Off"

This determines whether or not to record button press information in the log file. Normally set to "**Off**" unless a UI design study is being conducted. In this case, you would specify "**On**" so that you would know what UI control the subject used to interact with the simulation, eg., did they prefer popup context menus or push buttons.

VALID SET OF VALUES: ON or OFF

LogInfoInterface "on"

This determines whether or not to log information or action events, eg., the subject increased speed for target ABC, or targets ABC and DEF suffered a loss of separation. Should always be "**On**" unless you don't want to collect data about the run.

VALID SET OF VALUES: ON or OFF

3.1.2.3 Randomization Control

Seed 12345699

This parameter specifies the seed for random generation of application parameters (target parameters, airports...). Running the experiment with the same INI parameters (which specify the process of generation of random objects) will always have the target characteristics and entry points and airport configuration.

VALID SET OF VALUES: Any whole number greater than zero. Good seeds should be multi-digit and odd.

3.1.2.4 Clock Control

ClockDir "suicide"

Parameter specifying the trial clock direction. This should always be "**suicide**" for ATC runs.

VALID VALUES: "up' for clock counting up, "down" for clock counting down and "suicide" for clock counting down and terminating the program when the session time expires.

StartTime "210"

Initial time for the session clock (*in seconds*). Note that the *Ramp* target scheduler may not be able to run if this number is too low.

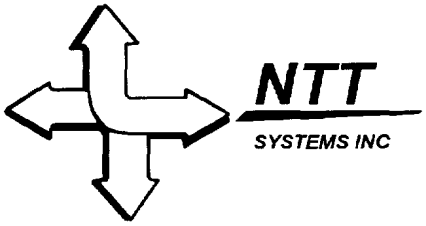
VALID SET OF VALUES: Any whole number greater than zero.

TrackTickRate "3"

This parameter specifies the length of the ATC update interval, in seconds.

VALID SET OF VALUES: Any whole number greater than zero.

ShowUpTime "2"



This is a number of time intervals that the target can be seen in the schedule window before it appears on the radar screen. Setting this to a number at least as large as StartTime will make all targets visible as soon as they are created.
VALID SET OF VALUES: Any positive whole number.

3.1.2.5 Near Miss Control

```
NEAR_MISS   "2     2000"
```

This parameter specifies the number of dots and distance in feet (exactly in that order) for near miss parameteres.
VALID SET OF VALUES: Positive whole numbers.

3.1.2.6 Target Schedule Control

```
!*************************************************************
!*    Target Generator Options
!*************************************************************
!             TYPE       NAC   %HV   %LT   %HE   #RAMPS
!SCHEDULE   "CONSTANT    5     50    40    10"
!SCHEDULE   "RAMP        5     50    30    20       4"

SCHEDULE    "Manual        C:/Atc/tgtScript/Atc.tgt"
```

Only one of the three options should be uncommented or the last uncommented option will be taken into consideration. Constant and RAMP schedules specify number of aircrafts (NAC), percent of heavy planes (%HV), percent of light planes (%LT) and percent of helicopters (%HE) in the simulations, respectively. In addition to that, the Ramp target loader specifies the number of ramps (RAMPS) that will be created in the session.
The Manual target schedule specifies only the name of the file that contains the aircraft descriptions. Manual schedules are discussed in Section 3.1.3.

3.1.2.7 Airport Control

```
!*************************************************************
!*    Airport Generator Options
!*************************************************************
AIRPORTS     "3"
```

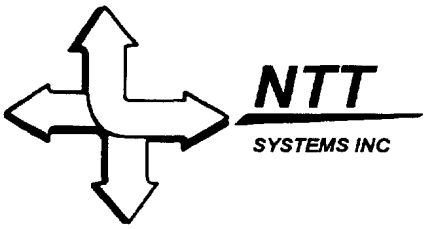
VALID SET OF VALUES: any whole number.

```
!AIRPORTS "20000/20000/45 30000/30000/90 -20000/-80000/180 -
50000/60000/270 40000/-50000/0"
```

VALID SET OF VALUES: any number of x/y/z pairs, as described below, ON THE SAME LINE!

The first option specifies only the number of airports that will be generated randomly. If AIPORTS is specified as zero or negative there will be no airports in the trial. If a manual target is specified as leaving from an airport that does not exist, it will appear at the centre of the screen at 1000 feet.

The second option explicitly sets the airports coordinates/runway in 'x/y/z' form. X is the number of dots from the radar center in horizontal direction; y is number of dots from the radar screen in vertical



direction; each ***multiplied by 10000***; and z is the runway course, with 0 indicating North, 45 North-East, 90 East, 135 South-East, 180 South, 225 South-West, 270 West and 315 North-West. ...

At least one of the options must be present and if both are uncommented then only the last one will be considered.

NoOfLandings "5"
This parameter specifies the maximum number of generated aircraft that can have airports as destinations. Ignored when manual target files are used.
VALID SET OF VALUES: Any positive whole number.

AutoLaunch "2"
This parameter specifies the number of intervals before a plane is launched automatically from an airport after its designated departure time.
VALID SET OF VALUES: Any positive whole number.

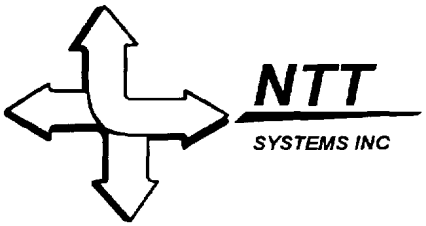
AutoIncreaseSpeed "true"
This parameter specifies if the speed of plane will be automatically increased to its maximum speed after takeoff.
VALID SET OF VALUES: 'true' for automatic increase of speed and 'false' for keeping the same speed.

3.1.2.8 Pause Control

```
!*************************************************************
!*    Experiment Pause Options
!*************************************************************
!        type    duration #ofpauses startDeadband endDeadband   inter-pause

!PAUSE "RANDOM     10          4            10          10              10"

!         type    duration startDeadband
!PAUSE "CONSTANT    10          10"

!          type  duration actual pauses
!PAUSE   "MANUAL  10         50/110/170"
```

If there is more than one of these options uncommented, then only the last one will be taken into consideration. All values are in seconds. If all are commented, or none is specified, there will be no automatic pauses in the trial. The trial clock is suspended when a pause is running. VALID SET OF VALUES:

- type: "RANDOM", "CONSTANT" and "MANUAL" (case insensitive)
- duration: any positive whole number
- #ofpauses any positive whole number
- start/endDeadband any positive whole number
- inter-pause any positive whole number



PAUSE_ENABLED "on"

This parameter specifies if 'Pause' button should be visible or not. If "On" the subject can initiate pauses at any time. The "Pause" button will become a "Continue" button until the subject clicks on it. At this point, the trial will resume. The trial clock is suspended during a pause.
VALID SET OF VALUES: "on" and "off" (case insensitive).

HideExpOnPause "on"

This parameter determines whether or not the ATC window is displayed during pauses. It will be hidden if "On" is specified.
VALID SET OF VALUES: "on" and "off" (case insensitive).

3.1.2.9 Questionnaire Control

Pause_Questionnaire "Off"

This parameter determines whether or not a questionnaire should start every time the trial is paused. Note that the trial will resume as soon as the questionnaire has been completed.
VALID SET OF VALUES: "on" and "off" (case insensitive).

QPeriod "0"
QPeriodEnd "off"

The first parameter specifies the number of seconds that will elapse between automatic presentations of the questionnaire. Zero or negative signals that the questionnaire should not be launched periodically during the run. Note that it will also be launched if the subject presses the "PAUSE" button and PAUSE_QUESTIONNAIRE is ON.

The second parameter indicates whether or not the questionnaire will be launched when the session time expires.

Both parameters are required.
VALID SET OF VALUES FOR QPeriod: any whole number.
VALID SET OF VALUES FOR QPeriodEnd: "on" and "off" (case insensitive).

TLX "C:/Atc/x.qst"

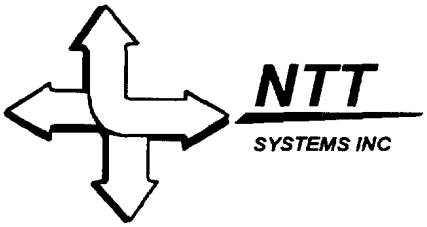
The name and location of the questionnaire file.

VALID SET OF VALUES: Any valid QST file. File paths must be specified with *forward* slashes. The questionnaire file implements a version of the NASA TLX and should not be modified.

3.1.3 The Optional Target File

The following is a sample target file:

```
"id Name Type Hspd Lspd Hupd Lupd Speed Altit Cours ExCours ExAltit EntLag CtlResid"
"%d  %s   %s   %f   %f   %d   %d    %f   %ld   %f      %f      %f      %d      %d"
1    ABC  Hv   240  120   1    2     120  5000    0      45     1000      1       20
2    XYZ  Lt   120   80   2    3      80  7000  270      90     2000      2       50
3    RVX  He    80    0   3 1000      80  1000   90     180     3000      2       55
4    DEF  Hv   240  120   1    2     120     3   -1      45     4000      7       30
5    GHI  Hv   240  120   1    2     120 10000  180      -1        2      9       45
```



The first two lines contain control information and must be coded *exactly* as shown, except that a tab can be substituted for sequences of spaces. Each aircraft is coded on a single line. The columns have the following semantics and allowable values.

`id`
This is the track ID that is recorded in the result file. It must be unique.

`Name`
This is the name of the target. Three characters, unique for the run.

`Type`
Must be Hv, Lt, or He, for heavy light or helicopter.

`Hspd`
The speed that is reported if travelling at high speed. Usually 240, 120, 80 (Hv, Lt, He)

`Lspd`
The speed that is reported if travelling at low speed. Usually 120, 80, 0 (Hv, Lt, He)

`Hupd`
The number of updates between moves at high speed. Usually 1, 2, 3 (Hv, Lt, He)

`Lupd`
The number of updates between moves at low speed. Usually 2, 3, a very large number (Hv, Lt, He)

`Speed`
Entry speed. Usually 240, 120, 80 (Hv. Lt, He)

`Altit`
Entry altitude. Should be a multiple of 1000 feet. Or, code airport number if departing from an airport.

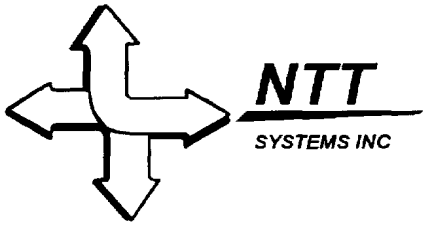
`Cours`
Entry course. 0, 45, 90, 135, 225, 270, or 305, where 0 is North. Or, code "-1" if departing from an airport.

`ExCours`
As Cours. Use "-1" if landing at an airport.

`ExAltit`
Exit altitude. Should be a multiple of 1000 feet. Or, coed airport number if landing at an airport.

`EntLag`
The update number at which the aircraft will enter or be available for takeoff.

`CtlResid`
The number of updates that can pass before the aircraft is considered to be late leaving the airspace.



3.2 Running ATC

ATC is typically run from the batch file AtcSoloWith.bat, eg., AtcSoloWith demo.ini. The batch file sets up the Java environment and starts the program passing it, in this case, demo.ini as the sole parameter. The first parameter, the name (and path) of the .INI file, is always required. As a convenience, you may specify any of the .INI file parameters on the command line. For example,

AtcSoloWith exp3.ini seed~'38241'

runs ATC with .INI file exp3.ini and uses 38241 as the value for the SEED parameter. Note that the value is enclosed in *single* quotes and is separated from the name by a tilde (~). This is useful for running multiple trials with the same conditions but with different randomization.

This feature significantly reduces the number of .INI files needed for an experiment. For example, assume that we want to hold all parameters constant. We would still need one .INI file for each trial since the .INI file specifies the log file name and header tag. To solve this problem, we could omit these parameters from the .INI file and move them to the command line, eg.,

AtcSoloWith exp3.ini header~'cond2S1' logfile~'c:/results/exp3/cond2S1.log'

3.3 Result Files

Result files consist of as set of ASCII records. All have a common header followed by event specific information. These fall into two categories, target events (TA records), and questionnaire results (QST records). All have a fixed format header consisting of four tab-delimited fields:

```
XCXCXCXC   20000509 11:45:59 EDT 00:00:09.94      Leader
|-------|  |------------------| |---------|      |----|
   (1)              (2)              (3)           (4)
```

(1) The value specified for the "Header" parameter in the .INI file.

(2) Date/Time that the event was logged.

(3) Value of the trial timer when the event was logged (usually decreasing for ATC).

(4) The literal text "Leader" (a TITAN holdover).

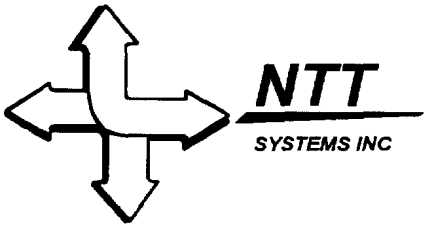
Following records are type specific and are shown in the following section.

3.3.1 Aircraft Events

Target entry event:

"TA" target_number "ACEN" target_name target_type target_course target_altitude target_exit_course target_exit_altitude

```
TA    1     ACEN  GSK   HV    SW    02    NW    03
```



Target exit event:
"TA" target_number "ACEX" target_name target_type target_course target_altitude target_exit_course target_exit_altitude

```
TA    1     ACEX  GAP   HV    W     05    W     05
```

Collision:
"TA" target1_number "COLL" target1_name target1_type target1_course target1_altitude target2_name target2_type target2_course target2_altitude

```
TA    16    COLL  XUF   LT    N     09    DPH   HE    S     09
```

Departure from airport
"TA" target_number "DFAP" target_name target_type delay

```
TA    14    DFAP  HV4   Hv    NW    01
```

Delayed autolaunch
"TA" target_number "DLAL" target_name target_type delay

```
TA    12    DLAL  EOT   LT    9
```

Delayed launch
"TA" target_number "DLCH" target_name target_type delay

```
TA    11    DLCH  HV1   Hv    3
```

Missed direction landing on the airport:
"TA" target_number "MDAP" target_name target_type "AP" airport_ID target_course target_altitude "AP" airport_ID airport_course "00"

```
TA    11    MDAP  JIG   LT    AP    01    N     00    AP    01    NE
00
```

Missed Direction - along a cardinal line
"TA" target_number "MDCD" target_name target_type target_course target_altitude target_exit_course target_exit_altitude

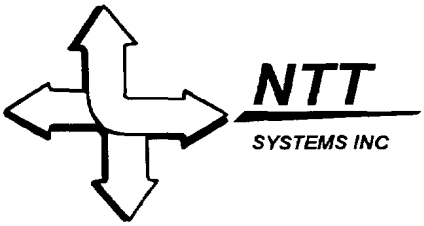
```
TA    3     MDCD  BTM   LT    SE    00    LT    AP    02
```

Missed Direction - not on cardinal line
"TA" target_number "MDNO" target_name target_type target_course target_altitude target_exit_course target_exit_altitude

```
TA    2     MDNO  JRQ   LT    S     09    LT    SE    09
```

Missed Approach Speed
"TA" target_number "MSAP" target_name target_type target_speed target_exit_course target_exit_altitude

```
TA    11    MSAP  JIG   LT    120   S     02
```



Near Miss Distance:
"TA" target_number "NMDT" target1_name target1_type target1_course target1_altitude target2_name target2_type target2_course target2_altitude number_of_dots "dot" distance_between_target1-target2 "feet"

```
TA    6     NMDT  OVY   LT    NW    09    JRQ   LT    S     09    2
dot   2000  feet
```

Premature Landing:
"TA" target_number "PMLD" target_name target_type target_course target_altitude target_exit_course target_exit_altitude

```
TA    3     PMLD  BTM   LT    SE    00    LT    AP    02
```

3.3.2 Subject Commands

These indicate that the subject attempted to modify aircraft characteristics. The final field in each record indicates whether or not the command was successful. It might be refused if, for example, the aircraft was at an airport and not ready to take off.

Change altitude:
"TA" target_number "AC" target_name target_type "SETALT" new_altitude "T"/"F"

```
TA    3     AC    BTM   LT    SETALT      03    T
```

Change course:
"TA" target_number "AC" target_name target_type "SETCRS" new_course "T"/"F"

```
TA    AC    BTM   LT    SETCRS      N     T
```

Change speed:
"TA" target_number "AC" target_name target_type "CHGSPD" "-1"/"1" "T"/"F" (-1 = decrement, 1=increment)

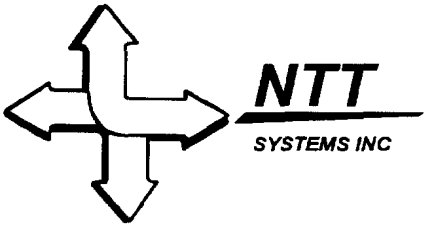
```
TA    AC    BTM   LT    CHGSPD      -1    T
```

3.3.3 Questionnaire Output

Each run of the questionnaire produces a single line of output. *Trial_number* indicates the presentation number (1 for the first time, 2 for the second...). *Qst_file* is the file name of the questionnaire. There are a pair of values for each question that was presented. The first is a tag that identifies the question. The second represents the subject's response. For sliders and radio buttons, this is a number from 0 to 100 (We treat radio buttons as evenly spaced points on a discrete slider). For regular buttons, the response value is the response tag that was set up in the questionnaire file.

"QST" trial_number quest_file_name tag/val tag/val ...

```
QST   1     C:/Atc/x.qst     Q1/74 Q2/19 Q3/20 Q4/10 Q5/80 Q6/80
```



4. System Overview

This version of ATC was developed in Java, making extensive use of existing components developed by *NTT* for the TITAN project. The system architecture is completely object oriented and relies extensively on the Model-View-Controller paradigm. The book, "Design Patterns," by Gamma et al., was used extensively as a guide.

In this approach, the underlying simulation consists of *model* elements that implement things like moving targets, radar systems, etc. Any time that a component changes state, it notifies its *observers*, or *views*. These are responsible for updating display components, for example, the radar view. Models have no knowledge of how their information is rendered, and are insensitive to the number of views that are observing them. This means that UI changes have absolutely no impact on the underlying simulation. In ATC for instance there are two views that reflect target information. The circular radar view simply watches the radar model and shows targets as icons moving across the display. The table view (schedule) observes a table model that in turn observes both the radar view and the target loader. This way it combines information about active and upcoming targets. The view shows a superset of the radar view's targets using tabular text. Changing this organization by adding or removing views, or changing a rendering strategy would have absolutely no effect on the models.

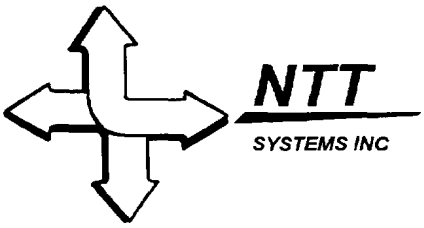
The control loop is closed by *controller* components. These are UI components that can modify model elements. For example the "+" speed button sends an "increment speed" command to the radar model. It finds the ID of the selected target and passes it, with the command to the world. The world in turn finds the target and tells it to increment its speed. The radar model observes that the target changes state (*if* it does) and notifies its views causing them to reflect the new information. Although the approach may seem round about, it accurately reflects the real world and provides obvious ways to implement enhancements like transmission errors, detection errors, more sophisticated movement or detection algorithms, etc.

The following sections are intended as a system overview and roadmap into the more detailed JavaDoc class documentation, provided in the *doc* directory in the distribution package.

4.1 Bootstrap Classes

ATC makes extensive use of TITAN classes and design strategies. For ATC we have built new loaders based on, but not *derived* from the TITAN LeaderLoader and SubordinateLoader classes. These are called AtcLeaderLoader and AtcSubordinateLoader. This approach lays the foundation for a multi-player ATC task. In the current configuration, all ATC runs behave like TITAN single player runs, and loading is controlled by AtcLeaderLoader. It reads the .INI file and uses AtcSubordinateLoader to parse the .GUI file. In doing this, it actually creates all of the model, view and controller components that comprise the task.

The GUI file is actually a list of classes. Each starts out with the class name and continues with parameters specific to each class. The loader uses Java's class Class to instantiate the named class and passes the rest of the information to its *config* method. In reading its parameters, it may find other class



names and the process recurs. In this way, for example, the TargetController (a view) contains the other UI components that comprise it. The entire system is created and assembled according to the .GUI script. This allows us to easily change what components are included, where and how large views and controls are, etc.

This flexibility is demonstrated most strongly with our button, command and action provider classes. Any control on the display, for example, increment speed or PAUSE, is simply a kind of *MenuItem.* When it is configured it receives the name of an instance of some kind of *Command.* When *Command*s are config'd they get a parameter and the name of an *ActionProvider.* An *ActionProvider* is simply an interface that obligates its implementer to provide a *doCommand* method.

When a button is clicked, it finds the named command object and runs its *execute* method. This in turn finds the named *Action Provider* and invokes its *doCommand* method, passing the parameter. The *ActionProvider* then performs the action specified by the parameter. A *Command* can be a *MacroCommand* that contains a number of commands, or a *CommandRouter* that can branch based on some named value. In this way, the system can be rewired without reprogramming the classes.

When objects go through their *config* method, they can be instructed to register themselves with the *NameManager.* This is a Singleton class that maps name strings to objects. If a command has "theRadarModel" as an *ActionProvider,* it doesn't really care what class is providing that function. Since the names are only bound to objects at run time, we have very loose coupling between our components. This allows us to mix and match components as needed.

By the time AtcSubordinateLoader has finished processing the .INI and .GUI files, the entire system has been constructed. It then locates and executes a command named "CMD_StartCommand" which, in the current implementation, does nothing. When the trial clock expires, it runs "CMD_EndCommand" and then exits. Note that in the .GUI supplied with the package, both of these are empty. The experiment is actually started when the subject hits the *StartPauseContinue* button and "CMD_StartExp" is run.

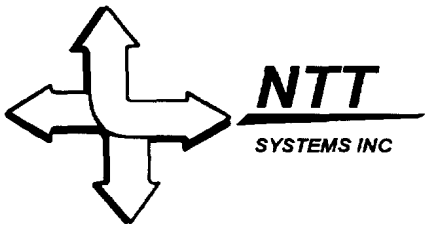
4.2 The Core Models

These models implement the underlying ATC simulation.

4.2.1 Moving Targets (ntt.atc.models.AtcMovingTarget)

Derived from ntt.titan.models.MovingTarget, this is basically a hashtable of target characteristic names and values. The primary differences are that the Atc MovingTarget has more complex movement rules, understands the semantics of several of its characteristics and most importantly, will act on changes to characteristics like speed, altitude and course.

Atc MovingTarget contains a static method, *getNextSpecs* that will randomly generate a set of characteristics according to a set of rules. All target loaders, except for the manual one, use this to create new targets as needed.



4.2.2 The World (ntt.atc.models.AtcWorld)

Derived from ntt.titan.models.World, AtcWorld controls the main clock and introduces time and space into the simulation. It accepts airports and targets from their respective loaders and acts as an *ActionProvider*, passing course, speed and altitude commands to targets. At the end of each period it tells all targets to update themselves and performs checks for near misses, collisions, landings and exits. These are all recorded when they occur. Updated targets are serialized and sent as proxies to the RadarModel.

4.2.3 The Radar Model (ntt.atc.models.AtcRadarModel)

Derived from ntt.titan.models.RadarModel, AtcRadarModel is capable of handling targets that can appear after the session starts and can exit before it completes. It also provides a mechanism for passing commands to modify target characteristics to AtcWorld for implementation.

AtcRadarModel replaces targets with updated ones when they arrive from AtcWorld. This is analogous to performing a radar scan. The new targets are proxies for the ones in the world. They can report their characteristics, but can not modify themselves in any way. When a scan has been completed, AtcRadarModel notifies its observers, thus propagating the information to the subject through the GUI.

AtcRadarModel also receives *hook* and *unhook* commands that designate the current target of interest. Observers are notified of changes in a target's selected status.

4.2.4 Airport Loaders

Airport loaders are created by ntt.atc.models.AirportLoaderFactory based on information in the AIRPORTS .INI parameter. The specified type of loader is created and sent to both AtcWorld and AtcRadarModel.

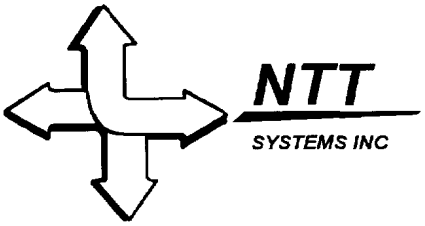
4.2.5 Target Loaders

As with airports, there are different types of target loaders (creators), one for each strategy. Once instantiated, the target loader is sent to AtcWorld who lets it know about the passage of time. When appropriate, the target loader will respond by creating new targets and inserting them into the world.

4.3 Core View/Controllers

4.3.1 The Radar View (ntt.atc.views.AtcRadarView)

This is a complete replacement for ntt.titan.views.RadarView. It is responsible for displaying the radar view background, targets and airports. The latter are rendered by AtcTrackView and AirportView. AtcRadarView observes AtcRadarModel, reflecting target changes to the subject. It also acts as a controller. The subject can click on targets, thus *selecting* them. This action is sent to the RadarModel, thus establishing a new default target for commands. When AtcRadarView is notified of the change in selection status, it refreshes the appropriate AtcTrackView's.



4.3.2 The Table Display View (ntt.atc.views.TableDisplayView)

This class is an awt.panel that contains a JTable, the only Swing component used in the application. It is a view of an ntt.atc.models.TrackTableModel. This in turn is an observer of both AtcWorld and the current target loader. As targets are created, arrive, change state and exit, they are added to, updated in, or removed from the table. It in turn notifies its sole observer, the TableDisplayView, which in turn renders the new information textually. Like AtcRadarView, it knows AtcRadarModel as an *ActionProvider*, and executes target selection commands when the subject clicks on a target in the table. This is an alternate way for the subject to indicate the target they are interested in operating on.

4.3.3 The Target Controller (ntt.atc.views.TargetController)

This class contains the indicators and controllers that allow the subject to send commands to the currently selected aircraft. It is in fact simply a configurable panel. In the .GUI script it is set up to contain two ControllerView's, one *round* and one *vertical*. These in turn each contain an Indicator and a set of buttons, CourseButton's for the former and AltitudeButton's for the latter. The Indicators each have a command that can fetch either course or altitude information. These are executed each time they observe the RadarModel selecting a new target. The returned information is used to modify the display. Each of the buttons is simply connected to a command that sends the appropriate altitude or course command to the AtcRadarModel.

In addition, the TargetController contains two speed buttons. These are also connected to commands, in this case passing "increment" or "decrement" commands.

4.4 Ancillary Components

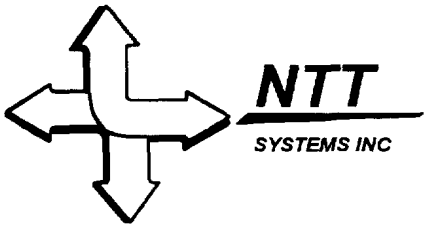
The ATC application required some functionality that was new and could not be derived from or based on existing TITAN classes. These included the requirements of having to automatically insert pauses during the trial and to automatically present the NASA TLX questionnaire periodically during and at the end of the trial.

Both of these requirements were conditional. The Experimenter had to be able to control these, so the solution is based on parameters in the .INI file and conditional execution of commands, controlled by CommandRouter's. We still had to develop classes to trigger the execution sequence and a whole package of classes to implement questionnaires. Doing the latter gave us a mechanism that could be used as a general questionnaire construction and execution mechanism.

4.4.1 Periodic Actions

PeriodicAction's are characterized by references to a clock, a start command and an end command. When their firing time comes it runs the first command. If the action is not for a questionnaire, it waits the designated time and then runs the end command.

If a questionnaire is run, *it* is responsible for cleaning up after itself and it has its own terminating command. This approach is needed because the questionnaire has its own set of GUI objects and thus its own thread. The thread that launches it returns immediately, leaving the display to be managed by Java's



event handlers. As a result, the launching PeriodicAction gets control back while the questionnaire is being displayed. This is why the questionnaire has to clean up after itself.

This design is flawed and shows its history, i.e., that PeriodicAction was designed to launch questionnaires and extended for pauses. A better redesign would have a base PeriodicAction with derived classes for pause actions and questionnaire actions.

4.4.2 Pause Management

Pause onset and duration could be specified manually or according to regular or controlled random rules. This is similar to airports and targets and we took the same approach. Class ntt.atc.util.PauseFactory creates the appropriate pause object according to the PAUSE parameter in the .INI file. The product is some kind of ntt.atc.util.PeriodicAction; specifically, ManualPeriodicAction, ConstantPeriodicAction or RandomPeriodicAction.

4.4.3 Questionnaires (ntt.util.question)

Questionnaires are created in much the same way as the ATC GUI. A file contains a list of classes that have to be instantiated. These include QuestionPage's that in turn can contain sliders, buttons, radio buttons, and question text. There can be single or multiple questions per page.

The questionnaire is managed by ntt.util.question.Questionnaire. It constructs the GUI and moves from page to page as the questions are answered. Questionnaire objects are instantiated because they appear in the .GUI file. There can be more than one, each tied to a questionnaire file by a parameter in the .INI file and referenced in the .GUI file. Each Questionnaire keeps track of how many times it has been run and this is recorded each time it produces its output. Output is sent to the standard data logger and appears in the result log.

DOCUMENT CONTROL DATA SHEET

1a. PERFORMING AGENCY NTT Systems Inc., 86 Dunblaine Ave., Toronto, ON M5M 2S1	2. SECURITY CLASSIFICATION UNCLASSIFIED -
1b. PUBLISHING AGENCY DCIEM	
3. TITLE (U) Air traffic control task - User guide and system overview	
4. AUTHORS Grushcow, M.	
5. DATE OF PUBLICATION March 31 , 2000	6. NO. OF PAGES 17
7. DESCRIPTIVE NOTES	
8. SPONSORING/MONITORING/CONTRACTING/TASKING AGENCY Sponsoring Agency: Monitoring Agency: Contracting Agency : Tasking Agency:	

9. ORIGINATORS DOCUMENT NUMBER Contract Report 2000-121	10. CONTRACT GRANT AND/OR PROJECT NO. W7711-8-7583/001/TOR	11. OTHER DOCUMENT NOS.

12. DOCUMENT RELEASABILITY Unlimited distribution
13. DOCUMENT ANNOUNCEMENT Unlimited

14. ABSTRACT

(U) The Air Traffic Control Task was originally developed by Somapro Ltd., in C for the Mac. This document describes a re-implementation of the task in Java, a language that runs on a variety of computer architectures, including PC/Windows, the DND standard. The goal was to create a functionally equivalent version of the task while redeveloping certain hard to use features
We are assuming that the reader is familiar with ATC. If that is not the case, please refer to the Somapro documentation This document provides detailed information on the following..
1. Describe how to configure and run the task and interpret its output.
2. Install the task on a Wintel computer.
3. Provide an overview of the software architecture and references to more detailed information.

15. KEYWORDS, DESCRIPTORS or IDENTIFIERS

(U) Air traffic control; simulation; human performance

#515029