SORTS Tech Report

James Irizarry, Sam Wintermute, Joseph Xu

July 25, 2006

0.1 ORTS Overview

The Open Real Time Strategy software is a highly configurable game engine used to play real time strategy (RTS) games [1]. The main purpose for ORTS is to serve as an open source, open interface RTS game engine for RTS AI tournaments. ORTS is undergoing active development as of July 2006 at the University of Alberta under the direction of Michael Buro.

There are several reasons why ORTS is especially suitable for use in AI tournaments. It has a (relatively) straightforward C++ API, making interfacing with your favorite AI system easy. All the specific game mechanics, ranging from types of units, actions, and physics, are specified via C++ style scripts called blueprints. This means that ORTS can be easily configured to simulate a wide range of environments, from arbitrarily simple ones like Wumpus World to complex ones like Starcraft. Finally, ORTS has a client/server architecture in which the server maintains the state of the world and only report to the clients information they are supposed to have for a fair game. This is in contrast to most commercial RTS games, in which each client maintains the entire world state and prevents the player from accessing forbidden information such as other players' locations only by hiding them from the GUI. The result is that ORTS is impervious to "memory hack" cheats that are widespread in commercial RTS games. This feature is particularly important if tournaments are to be run across the Internet.

0.2 SORTS Overview

SORTS is a piece of software that allows the Soar cognitive architecture to act as a client to the ORTS game server, so that ORTS game playing agents can be written in and executed on Soar. SORTS is much more than an interface bridge in that it intentionally constrains the space of possible Soar agents in important ways and also handles aspects of low-level game control that are not suitable for Soar. In particular, SORTS heavily abstracts the world state information obtained from the ORTS API before feeding them into Soar as perceptions, and also interprets and executes high-level commands from Soar as low-level actions sent to the ORTS API.

0.3 Updating the Game State

The ORTS API provides game state updates in the form of 6 lists:

- `new_tile_indexes` New game tiles encountered via uncovering of fog of war.
- `new_objs` New game objects, such as enemy units and buildings, or world objects such as trees.
- `changed_objs` Game objects that have changed since the previous viewframe.
- `vanished_objs` Game objects that disappeared from the player's vision.
- `dead_objs` Game objects that died.
- `new_boundaries` New terrain boundaries that demarcate the border between different terrain types, such as ground and cliff.

SORTS currently takes into account the information in all the lists except `new_tile_indexes` and `new_boundaries`. The former is ignored because SORTS currently does not have any functioning capability

```
ORTSEventHandler
  lockMutex()
  mergeChanges(oldChanges, changes)
  if currentViewFrame - lastActionFrame > ALLOWED_LAG then
    unlockMutex()
    return
  end
  removeDeadObjects(changes)
  assignSoarActions()
  updateSoarGameObjects(changes)
  sendActions()
  updateGroups()
  unlockMutex()
end
```

Figure 1: Pseudo code for the ORTS event handler

to reason about terrain, except for collision detection. The latter is ignored because we get this information from the game tiles directly whenever we check for boundaries. Note also that currently, SORTS treats objects vanishing (reported in `vanished_objs`) and objects dying (reported in `dead_objs`) as one and the same thing. This will probably change in the future as agents use more sophisticated reasoning.

All game state updates, both internal to the middleware and to the Soar input link, are performed in the ORTS event handler, which is triggered everytime the middleware receives a new viewframe from the ORTS server. The event handler is described below.

0.3.1 The ORTS Event Handler

The ORTS event handler is triggered everytime the middleware receives a new viewframe from the ORTS server. This event handler and the Soar event handler are the main functions from which most other function calls are made. Pseudo-code for the handler is shown in figure 1.

Each function call is described below.

- `lockMutex()` and `unlockMutex()` This mutex locks the Soar event handler from executing until the ORTS event handler is finished. This is important since we don't want the Soar command buffer to change while we are processing that buffer.
- `mergeChanges(oldChanges, changes)` If SORTS falls behind the ORTS server too much, changes to the game state will be accumulated but not processed in the interest of catching up. This function merges changes reported in previous viewframes with the current changes.
- `removeDeadObjects(changes)` This function removes all game objects that have just died or vanished in the current frame. This call needs to be made before the call to `assignSoarActions()` so that we don't try to assign any actions to objects that disappeared, which the ORTS server would not understand.

```
SoarEventHandler
  lockMutex()
  if Catchup = true then
    unlockMutex()
    return
  end if
  getNewSoarOutput()
  processVisionCommands()
  processGameCommands()
  unlockMutex()
end
```

Figure 2: Pseudo code for the Soar event handler

- `assignSoarActions()` Interprets commands queued from the Soar output-link as ORTS actions and distributes them to the appropriate groups. The actions are not sent to the ORTS server until `sendActions()` is called.
- `updateSoarGameObjects(changes)` This is the main function that updates the attributes of the SoarGameObject structures in the middleware.
- `sendActions()` This is a call to the ORTS API that sends all queued actions to the server.
- `updateGroups()` Runs the grouping algorithm over SoarGameObjects that have changed or appeared, and prunes those groups whose members have died. These changes are directly reflected in the Soar input-link, but are buffered in the SML interface until the next Soar decision cycle.

0.3.2 The Soar Event Handler

The Soar event handler is triggered at the end of each Soar decision cycle. The main responsibility of this function is to take commands off the Soar output-link and buffer them into action queues in the middleware.

Figure 2 shows the pseudo-code for the function. The following is a list of descriptions for each call made in the function.

- `lockMutex()` and `unlockMutex()` These calls lock on the same mutex as the ORTS event handler, thereby making the execution of these two functions mutually exclusive.
- `getNewSoarOutput()` Takes all commands on the Soar output-link and queues them into lists in the middleware. This includes both commands issued to in-game units as well as commands that adjust middleware parameters such as grouping radius and center of visual attention. However, none of the commands are actually processed by this function.
- `processVisionCommands()` Processes Soar commands related to the perceptual system. These commands include changing the grouping radius, looking at a specific coordinate, looking at a feature in the feature map, and changing the maximum number of objects allowed on the input-link at any time (for a complete list, see section 0.4.2. These commands must be processed here rather than with in-game commands (which are processed in the ORTS event handler) because Soar needs to have feedback from

the perceptual changes it requests in the very next decision cycle. Any kind of lag may result in strange behavior.

- `processGameCommands()` Handles all commands that affect the way the middleware executes low-level actions, such as in the FSMs, but do not translate directly into ORTS game object actions. Also handles miscellaneous queries that Soar may make to the middleware for information not normally provided to it. Currently, there are only three such commands: finding a location for a building, and setting and clearing the mineral buffer (see section 0.4.2).

0.4 Soar IO Description

0.4.1 The SORTS input-link

There are five top-level attributes on the SORTS input link, "groups", "game-info", "feature-maps", "vision-info", and "query-results". The groups, feature-maps, and vision-info structures are all part of the main visual system (see XXX), while game-info contains higher-level information about the game world, and query-results is used to communicate the results of specialized queries from Soar to the middleware.

The exact data structures are as follows:

Attributes of io.input-link	
attribute	description
vision-info structure:	
vision-info	Contains information on the current state of the vision system.
vision-info.center-x vision-info.center-y	The coordinate of the center of the region in view.
vision-info.focus-x vision-info.focus-y	The coordinate of the center of focus (spotlight of attention).
vision-info.num-objects-visible	The maximum number of objects (groups) present on the input-link. All other objects within the view window are present in feature maps.
vision-info.grouping-radius	All objects of the same type (except as below) and owner within this distance of each other are in the same group (set to 0 for individuals).
vision-info.owner-grouping	Ignore type when grouping, only group by owner (1 if enabled, 0 if disabled).
groups structure:	
groups	The set of groups being attended to.
groups.group	Multi-valued, one instance for each group. Detailed below.
feature-maps structure:	
feature-maps	Contains all feature maps- low-resolution information about certain features of unattended (but visible) objects.
feature-maps.friendly	Friendly feature map- each friendly unit (not group) results in one instance of this feature, marked in the sector of the group's center of gravity.
feature-maps.friendly-workers	A subset of the friendly feature map, showing only workers.
feature-maps.enemy	Similar to the above, but for enemy units.
feature-maps.minerals	Similar to the above, but for minerals.
feature-maps.moving-units	Moving units (if grouping is used, one moving object causes the whole group to be seen as moving).
feature-maps.(any).sector0 .. feature-maps.(any).sector8	Feature counts for each of the nine sectors. 0 is the upper left, 8 is the lower right.
game-info structure:	
game-info	General information about the game (non-visual information).
game-info.num-players	The number of players.
game-info.player-id	The ID number of the Soar player.
game-info.map-xdim game-info.map-ydim	The dimensions of the game map.
game-info.view-frame	The last view frame handled by the middleware (the number of game cycles executed).
game-info.my-minerals	Minerals available to the player.
game-info.mineral-buffer	The number of minerals to reserve for certain tasks (see section XXX).
game-info.worker-count game-info.marine-count game-info.tank-count	The number of units of each type owned by the player.
query-results structure:	
query-results	The result of the last query to the middleware.
query-results.query-name	The name of the last query executed.
query-results.param0 query-resutls.param1	Query return values- the meaning of these is dependant on the query-name.

Attributes of io.input-link.groups.group objects		
attribute	which groups	description
num-members	all	The number of individuals comprising the group.
type	all	The type of the group (ex: worker, mineral).
x-pos y-pos	all	The x,y location of the center of gravity of the group.
x-min x-max y-min y-max	all	The bounding box of the group.
health	all	The sum of the health of all units in the group.
taking-damage	all	The number of members of the group currently taking damage (under attack).
shooting	all	The number of members of the group currently attacking an enemy.
speed	all	The average speed of the group.
heading	all	The average heading of the group.
dist-to-focus	all	The distance from the center of gravity of the group to the attentional focus point.
dist-to-query	all	The distance from the center of gravity of the group to the last query location.
owner	all	The player number of the group's owner.
enemy	all	1 if the group belongs to an enemy player, 0 otherwise.
sticky	friendly	1 if the group is sticky- sticky groups remain together even if they are no longer spatially close.
command	friendly	The last command issued to the group ("none" if no command has been issued).
command-running	friendly	The number of members of the group currently executing a command.
command-success	friendly	The number of members of the group that successfully completed the last command.
command-failure	friendly	The number of members of the group that unsuccessfully completed the last command.
minerals	friendly workers	The total number of minerals possessed by the workers in the group.
active-mining	friendly workers	The number of workers that are actively mining.

0.4.2 The SORTS output-link

The output-link allows the Soar agent to act in the game world by issuing commands to groups of friendly units, communicate directly to the middleware for actions such as querying, and adjust the parameters of the visual subsystem in the middleware. All structures on the output link are similar- the Soar agent creates an output-link.command object with a name (**output-link.command.name**) specified. Depending on the command name, different parameters are needed. Parameters are attached to the command structure, as in **output-link.command.param0**. All legal commands and their necessary parameters are detailed below.

Soar output-link commands		
command name	parameters	description
Vision commands		
grouping-radius	*value*	Change the grouping radius of the vision system.
enable-owner-grouping	(none)	Enable grouping-by-owner.
disable-owner-grouping	(none)	Disable grouping-by-owner.
change-view-width	*value*	Change the width of the agent's view to be *value*.
look-at-location	x, y	Move the focus of attention to the given coordinate (which must be in the current view window.
move-to-location	x, y	Shift the view window to be centered at the given coordinate, and make that the focus of attention.
look-at-feature	*feature*, *sector*	Move the focus of attention to a given feature in a given sector. The legal features are the names of the feature-map objects on the input link.
move-to-feature	*feature*, *sector*	As above, but re-center the view window to the new location, also.
num-objects	*value*	Change the maximum number of objects (groups) present on the input-link at once.
General middleware commands		
locate-building	*building*, x, y, *distance*	This command requests that the middleware attempt to find a location for a building of type *building* approximately *distance* units from the coordinate given. The resulting coordinate is returned through the query-result structure on the input-link.
increase-mineral-buffer	*value*	Increase the mineral buffer by *value* minerals.
clear-mineral-buffer	(none)	Set the mineral buffer to 0.
Group commands		
move	*group*0, *param*0, *param*1	Move the group with ID *group*0 to the x,y location where *param*0 is x and *param*1 is y, using the default precision of 10 (allow units to complete successfully if they are within 10 of the target).
move	*group*0, *param*0, *param*1, *param*2	Move group *group*0 to the x,y location where *param*0 is x and *param*1 is y, using a precision of *param*2.
build	*group*0, *param*0..*param*3	Use group *group*0 to build a building of type *param*0[a] at the x,y location given by *param*1, *param*2. If *param*3 is 1, the mineral buffer is used, otherwise it is not.
train	*group*0, *param*0..*param*2	Use group *group*0 (a building) to train units of type *param*1[b]. Train up to *param*2 units, and use the mineral buffer if *param*3 is 1.
attack	*group*0, *group*1	Use the friendly group with ID *group*0 to attack *group*1.
mine	*group*0	Assign *group*0 to mine minerals. The control center and mineral patch to use are automatically determined in the middleware.
stick	*group*0	Set *group*0's to be sticky- ensure it stays as one group, even if the members move apart. Assigning any of the above actions to a group does this by default.
free	*group*0	Clear *group*0's sticky status- allow the members to be split up and join other groups.
join	*group*0, *group*1	Force *group*0's members to join *group*1. If *group*1 is not sticky, this may be automatically undone the next cycle.

[a] Legal building types: 0=controlCenter, 1=barracks, 2=factory

[b] Legal unit types: 0=worker, 1=marine, 2=tank

Bibliography

[1] Michael Buro. Orts: A hack-free rts game environment. In *Proceedings of the International Computers and Games Conference 2002*, 2002.