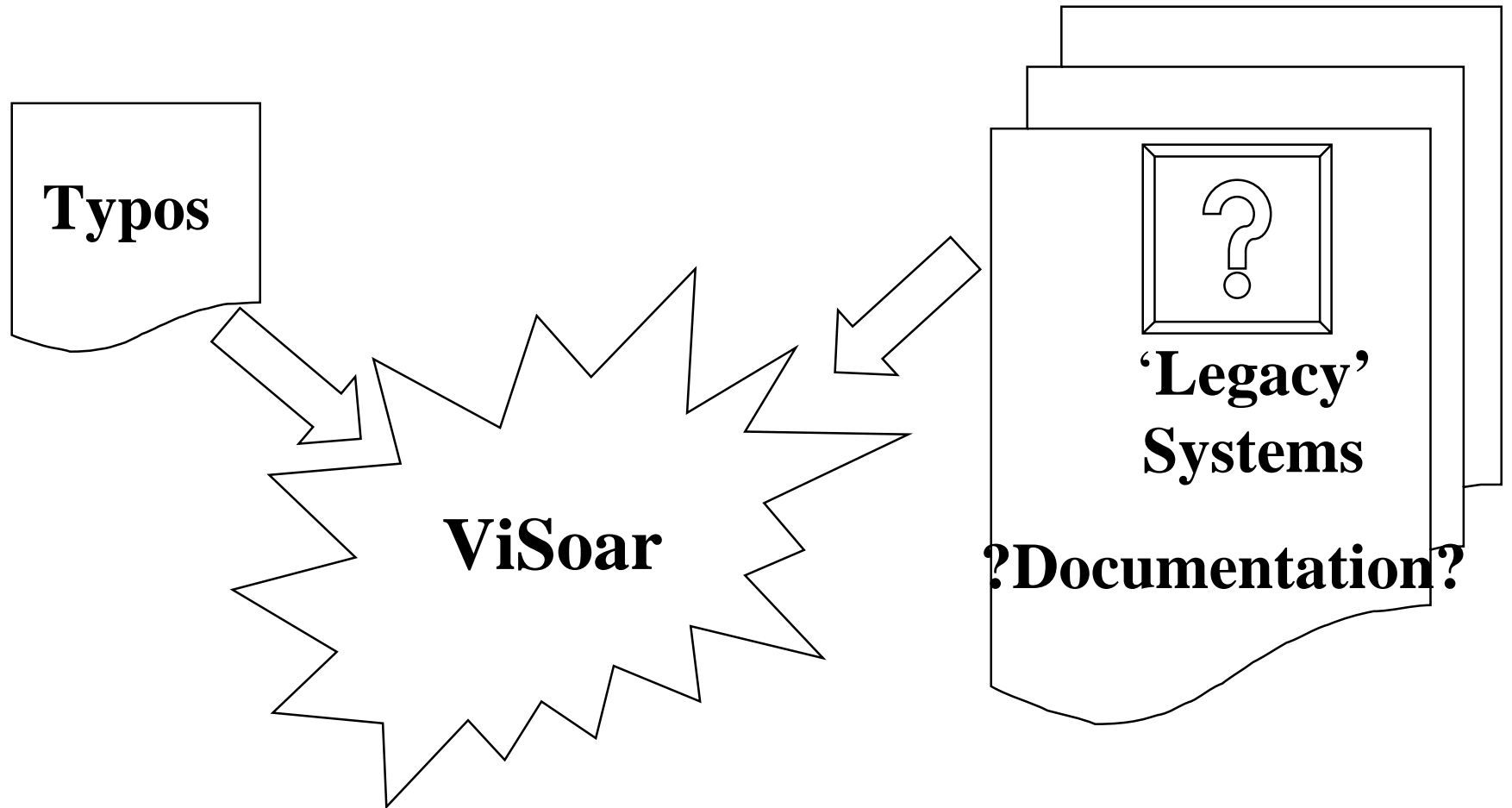


# ViSoar

Intelligent Agents Group,  
University of Portsmouth

<http://www.dcs.port.ac.uk/~hirsta/visoarx.htm>

# ViSoar Genesis



# Background to ViSoar

- Typos
- Whilst evaluating Milind's STEAM ruleset:
  - problems were encountered in setting up initial team states correctly
  - a large number of reusable components and attribute-value structures were identified
- This suggested the need for:
  - an automatic team generator
  - a 'reusable code exploiter'

# What is ViSoar?

- An integrated Visual Soar development environment.
- Currently comes in two parts:
  - ☆ – code generation tools (*deViSoar*)
  - ☆ – offline ‘reverse engineering’ tools (*reViSoar*)
    - automated debugger (*adViSoar*)
- Compatible with automatic high level knowledge representation language to Soar code translator package (*Soarceror*)

# Philosophy

- Visual environment
- Written in Tcl/Tk
- Exploit Soar architecture at command rather than code level
- Facilitate the construction of typo free Soar code

# deViSoar

- Text editor
- ‘*avtree* ontology’ editor
- Automatically generated ‘skeleton’ productions
- Various Soar code manipulating tools
  - OR
  - NOT

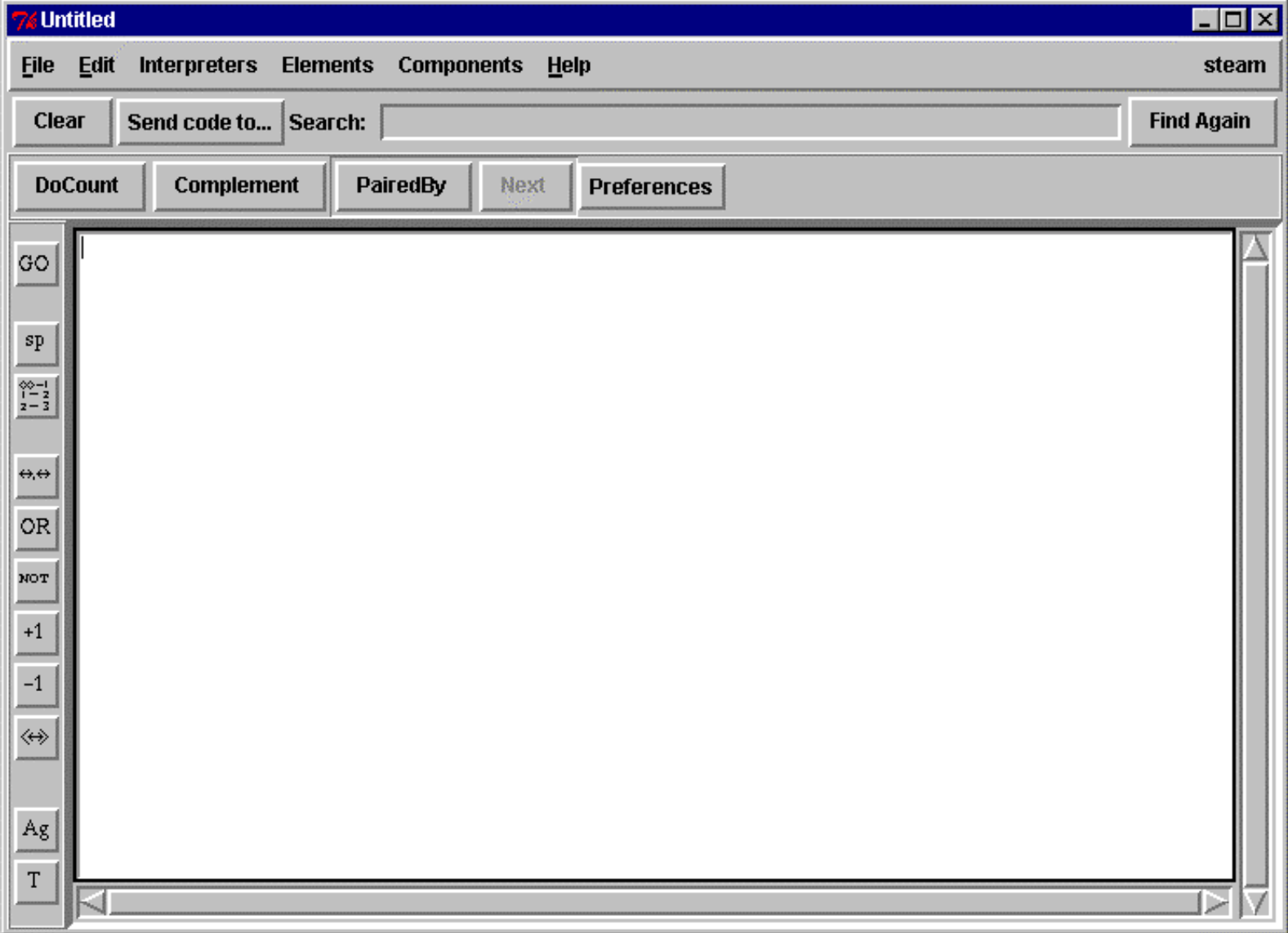
## *reViSoar*

- ‘Reverse engineering’ of legacy Soar agents
- Utilises an offline operator hierarchy analyser (*OpHelia*)
  - extracts implicit structure of rulesets
    - operator/problem space hierarchy
    - productions related to particular operators/problem spaces
- Generates *avtree* that may be passed to *deViSoar*

# Additional Components

- Production viewer (*ProdView*)
  - allows straightforward inspection of (large numbers of ) productions
  - available as standalone package with connector to TSI
  - *ViSoar* integrated version provides filtered searching of loaded productions





# deViSoar Menubar

File   Edit   Interpreters   Elements   Components   Help

- Standard *File* options (*Load, Save, etc.*)
- Standard *Edit* options (*Cut, Copy, etc.*)
- *Interpreters* raises *reViSoar* window for selected (or newly *Created*) Soar interpreter
- *Elements* contains project dependent *avtrees* and operator hierarchies
- *Components* contains toolbar commands and raises *Skeleton* production dialogues

# *deViSoar* Vertical Toolbar



- *GO* to relevant part of *avtree*
- Raise *core-lhs skeleton* dialogue
- Generate standard form list
- Toggle dot/expanded form
- Insert *or* construct
- Insert NEQ construct
- Increment indirector count
- Decrement indirector count
- Toggle indirector/constant

# Core-lhs Skeleton (1)

The image shows a dialog box titled "coresp\_lhs" with a blue header bar. The dialog contains several controls for configuring the generation of Soar code:

- State elaborations:** A label followed by a "Clear" button.
- Name production?:** A label followed by two radio buttons, "Yes" (selected) and "No".
- Feature Selection:** A row of six checkboxes: "operator", "problem-space", "top-state", "superstate", "input-link", and "output-link".
- Operator Elaborations:** A label followed by a checkbox and a text field containing "<v\_opname>".
- Problem-space Elaborations:** A label followed by a checkbox and a text field containing "<v\_psname>".
- Buttons:** "Generate Soar code..." and "Close" are located at the bottom of the dialog.

# Core-lhs Skeleton (2)

Automatically generated Soar code:

```
#####  
sp { apply*wait  
  (state <s> ^operator <v_o> ^io <v_io>)  
  (<v_io> ^input-link <v_ip>)  
  (<v_o> ^name wait )  
  
  -->  
  
}
```

# The *Operator Proposal* Skeleton

```
#####  
sp { *propose*operator*test_operator  
  (state <s> ^problem-space <v_ps> ^top-state <v_ts> ^superstate  
    <v_ss> ^io <v_io> ^io <v_io>)  
  (<v_io> ^input-link <v_ip>)  
  (<v_io> ^output-link <v_op>)  
  (<v_ps> ^name <v_psname> )  
  -->  
  (<s> ^operator <v_o> + &)  
  (<v_o> ^name test_operator + )  
}
```

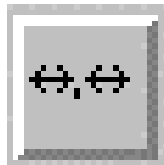
# The *problem-space* Proposal

```
#####  
sp { *create*problem-space*test_ps  
  (state <s> ^operator <v_o> ^top-state <v_ts> ^superstate <v_ss>  
    ^io <v_io> ^io <v_io>)  
  (<v_io> ^input-link <v_ip>)  
  (<v_io> ^output-link <v_op>)  
  (<v_o> ^name <v_opname> )  
  
  -->  
  (<s> ^problem-space <v_ps> + & )  
  (<v_ps> ^name test_ps + )  
  
}
```

# Exploiting dot notation

*Expand*

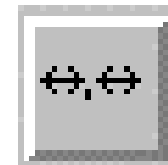
(**<s>** ^**grandparent.parent.child**  
grandchild)



(**<s>** ^grandparent **<vs6>**)  
(**<vs6>** ^parent.child grandchild)

*Collapse*

...^parent **<ind>**)  
(**<x>** ^**however many**)  
(**<y>** ^**infill lines**)  
(**<ind>** ^child val...)



...^parent.child val...)  
(**<x>** ^however many)  
(**<y>** ^infill lines)



# The *OR* Construct

(<s> ^choices <v\_ch>)



*either:*

(<s> ^choices { << multiple none >> <v\_ch> })

*or:*

(<s> ^choices { << >> <v\_ch> })

depending on user-preference.

# Inequalities

( $\langle v_o \rangle^{\text{name wait}}$ )

NOT

( $\langle v_o \rangle^{\text{name } \{ \langle \rangle \text{ wait } \}}$ )

Alternatively:

( $\langle v_{io} \rangle^{\text{input-link } \langle v_{ip} \rangle}$ )

NOT

( $\langle v_{io} \rangle^{\text{input-link } \{ \langle \rangle \langle v_{ip} \rangle \langle v_{ip}^*1 \rangle \}}$ )

# The *avtree* I - Implicit vs. Explicit Problem Spaces

- Explicit problem-spaces are identified with the *^problem-space.name* attribute
- Implicitly defined problem-spaces reflect the state elements manipulated by productions acting within the ‘problem-space’
- The *avtree* for productions whose behaviour we associate with activity in a given problem-space defines the problem-space

# The *avtree* II - Example

```
(state <s> ^lev1*a (<lev1*a> ^lev2 <lev2>) ^lev1*b val1)
(<lev2> ^lev3*a << val3a val3b >> -^lev3*b <x>)
-->
(<lev2> ^lev3*b val3c + &)
```

```
lev1*a
  lev2
    lev3*a
      val3a, val3b
    lev3*b
      val3c
  lev1*b
    val1
```

# The *avtree* III - Element Types

- The *avtree* for a ruleset essentially provides a hierarchically structured ontology:
  - major attributes (have attributes as values)
  - minor attributes (have constant values)
  - values
  
  - single attributes
  - multi-attributes

# Growing the *avtree*

- Attributes and values may be added to any *avtree* at any time from the *Elements* menu.
- *Add ^attr <ind>...* allows you to add a *major attribute* (an attribute that will have another attribute as its child)
- *Add ^attr const...* allows you to add a *minor attribute* (an attribute that takes a constant value), and will then prompt for its values
- *Add value...* allows you to a constant value
- Deleting *avtree* elements is currently unsupported.

# *avtree* Summary

The *avtree*:

- represents an ontology for a given Soar agent
- may be used as an implicit problem space definition
- may be created/added to, saved and loaded within *deViSoar*
- may be extracted from a legacy ruleset within reViSoar, and saved for use in *deViSoar*
- may be used for sophisticated search routines

*Add ^attr const...*

Prompts for attributes:



followed by appropriate prompts for values:



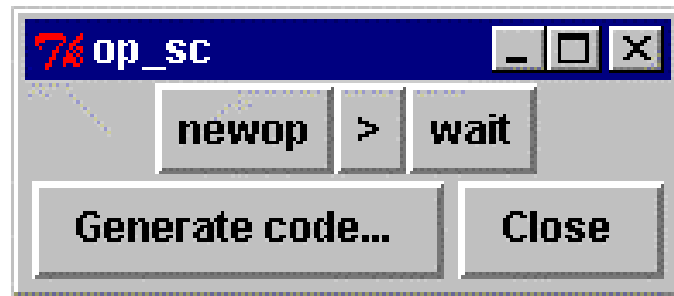


# Value Adding Shortcuts

- HML provides: \*high\* \*medium\* \*low\*
- YN provides: \*yes\* \*no\*
- NEWS provides: \*north\* \*south\* \*east\* \*west\*
- UD provides: \*up\* \*down\*
- RL provides: \*right\* \*left\*

Adding new shortcuts is straightforward.

# Operator Search Control



```
#####  
sp {*search-control  
  (state <s> )  
  (<s> ^operator <op1> + <op2> + )  
  (<op1> ^name newop )  
  (<op2> ^name wait )  
  
  -->  
  (<s> ^operator <op1> > <op2> )  
}
```

# Generating lists



will generate the list:

( <list\*0> ^item one ^next <list\*1> )

( <list\*1> ^item two ^next <list\*2> )

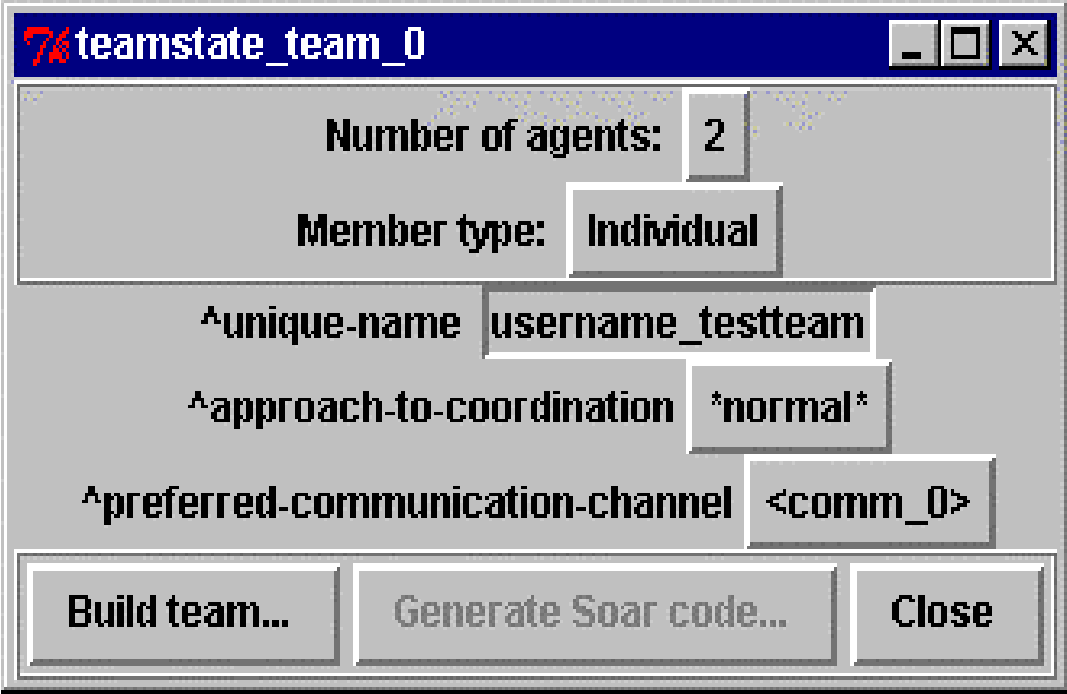
( <list\*2> ^item three )

## Add Ons for *deViSoar*

- deViSoar is intended to support the use of task specific skeletons
- Currently, several skeletons for use with Tambe's STEAM ruleset are provided
- Ideally, a user should be able to quickly and easily create new skeletons relevant to a given project

# STEAM Add-Ons: Team Creation 1

- Using the  button raises:



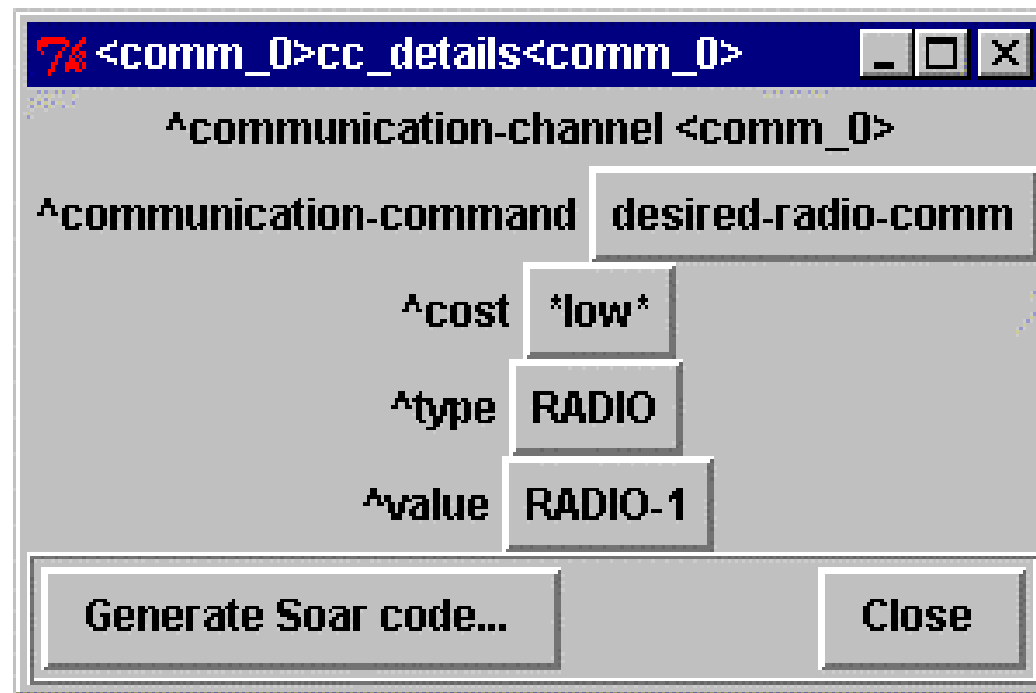
The screenshot shows a dialog box titled "teamstate\_team\_0" with the following configuration options:

- Number of agents: 2
- Member type: Individual
- ^unique-name: username\_testteam
- ^approach-to-coordination: \*normal\*
- ^preferred-communication-channel: <comm\_0>

At the bottom of the dialog, there are three buttons: "Build team...", "Generate Soar code...", and "Close".

# STEAM Add-Ons: Team Creation 2

- Raising the ‘communication channel’ menu provides an option to change channel properties:



# STEAM Add-Ons: Team Creation 3

- After ‘building’ the team, team definition code may be automatically generated (formatting sacrificed for clarity(!)):

```
sp {elaborate*teamstate*username_testteam
  (state <s> ^name top-ps ^superstate nil ) -->
  (<s> ^command <c> ) (<c> ^group <team_0> )
  (<team_0> ^unique-name username_testteam ^communicated <comm> ^colocated *no* )
  (<team_0> ^teamtype *yes* ^team-plan <tp> ^approach-to-coordination *normal*)
  (<team_0> ^member-list <m> ^team-leader username_testteam_agent_1 )
  (<m> ^leader <agent_1> ^member <agent_1> + & ^member <agent_2> + &)
  (<agent_1> ^unique-name username_testteam_agent_1)
  (<agent_2> ^unique-name username_testteam_agent_2)
  (<team_0> ^speaking-order <so> )
  (<so> ^username_testteam_agent_1 <so0> ^username_testteam_agent_2 <so1> )
  (<so> ^member username_testteam_agent_1 ^next <so0> )
  (<so0> ^member username_testteam_agent_1 ^next <so1> )
  (<so1> ^member username_testteam_agent_2 ^next nil )
  (<team_0> ^preferred-communication-channel <comm_0> ^communication-channel <comm_0> )
  (<comm_0> ^communication-command desired-radio-comm ^cost *low* ^type RADIO ^value
  RADIO-1)}
```

# STEAM Add-Ons: Team Creation 4

- when team definition code is generated, an option is raised automatically that will generate and save code for each agent, if required, along the lines of:

```
sp {elaborate*teamstate*username_testteam_agent_1
  (state <s> ^name top-ps ^superstate nil ^command <c> )
  -->
  (<s> ^self <agent_1> )
  (<c> ^group <agent_1> )
  (<agent_1> ^unique-name username_testteam_agent_1)
  (<agent_1> ^approach-to-coordination *normal*)
  (<agent_1> ^type individual)
  (<s> ^trust username_testteam_agent_1 + &
    ^trust username_testteam_agent_2 + &)}

```



# STEAM Add-Ons: Agent Creation 1

- The  button raises the dialogue:



The dialog box, titled "teamstate\_agent\_3", contains the following configuration options:

- ^unique-name**: username\_agent\_3
- ^approach-to-coordination**: \*normal\*
- ^type**: individual
- ^preferred-communication-channel**: <comm\_0>
- ^member-list.**: member
- ^team-leader**: none

At the bottom of the dialog are two buttons: "Generate Soar code..." and "Close".

# STEAM Add-Ons: Agent Creation 2

- If the agent type is ‘individual’, code will be generated of the form:

```
sp {elaborate*teamstate*username_agent_3
  (state <s> )
  (<s> ^name top-ps ^superstate nil )
  (<s> ^command <c> )
  -->
  (<s> ^self <agent_3> )
  (<c> ^group <agent_3> )
  (<agent_3> ^unique-name username_agent_3)
  (<agent_3> ^approach-to-coordination *normal*)
  (<agent_3> ^type individual)
}
```

# STEAM Add-Ons: Agent Creation 3

- If the agent is of type 'team':

```
sp {elaborate*teamstate*username_agent_3
  (state <s> ^name top-ps ^superstate nil ^command <c> )
  -->
  (<s> ^self <agent_3> )
  (<c> ^group <agent_3> )
  (<agent_3> ^unique-name username_agent_3 ^type team )
  (<agent_3> ^approach-to-coordination *normal*
    ^preferred-communication-channel <comm_0>)
  (<agent_3> ^member-list <agent_3_member>)
  (<agent_3_member> ^member <none> + &)
  (<agent_3_member> ^leader <none> ^speaking-order <so> )
  (<agent_3> ^communication-channel <comm_0> )
  (<comm_0> ^communication-command desired-radio-comm ^cost *low*)
  (<comm_0> ^type RADIO ^value RADIO-1)}
```

- Members may be added within the dialogue from a list of current agents (indls or teams) and suitable code will be generated

# *ViSoar* availability

*ViSoar* is available on an AS IS basis from:  
<http://www.dcs.port.ac.uk/~hirsta/tcltools.htm>

**Also, e-mail for details of the latest version of the manual**

*Please register...*