Languages and programming systems for robotics

- Introduction
- Requirements of robots programming
- Robots programming approaches
- Categories of programming languages
- Investigation of some systems / programming languages of robots.

Introduction

- Proposition of languages according to the programming methods and abstractions provided by the architectures

Requirements of robot programming

- Detection
- World modelling
- Motion specification
- Control flow
- Programming environment

Detection

- Use of position control without detection in most early industrial applications: World modelling
  - Environment designed to eliminate all significant sources of uncertainty.
  - Significant investment in design and specific equipment for each new application.
- The sensors make it possible to robots to perform tasks in the presence of significant environmental uncertainties without the need for specific tools

Main uses of sensors

- Launching and stopping of actions.
- Choice between different alternative actions.
- Obtaining the identity and position of objects and their supplies.
- Adaptation of the robot according to the external constraints.

Adaptive motion

- Active adaptation: Sensory interaction needed in situations requiring continuous movement in response to continuous sensory input.
- Operation specific to robotics.
- Difference in comparison to the first 3 uses of detection: extensibility.
  - Addition of new sensors and modules easy in the other cases (semantics determined only by the user program).
  - Strong integration between the sensor and the motion control subsystem for adaptive motion.

Programming mechanisms indirectly linked to detection

- Target positions not known during programming: possible to obtain from an external database or a vision sensor or simply defined by bumping into an object.
- Actual path to follow unknown at the time of programming: can be determined from the history of sensory inputs.
- Motion sequence unknown at the time of programming: execution sequence determined by the detection operations.

World modelling

- The data manipulated by robotic programs are primarily geometric.
- Need of means for :
  - the representation of the positions and supplies of objects such as surfaces and holes.
  - facilitate the calculation on the position of the objects and the configurations of the robot (representation of matrices, various operations on the matrices, etc.).
  - the description of the constraints existing between the positions (for example for the displacement of rigid objects.

Example: grasping an object

- Need to specify the desired position of the robot gripper relative to that of the object.
- Actual position of the object determined during execution using a vision system or an online database.
- Possibility of determining the position of the gripper by composition between relative positions of seizure and absolute position of the object.
- Transformation of this position into a configuration of the robot.

Motion specification

- Specification of final positions insufficient most of the time (e.g. risk of collision)

  => Need to specify a path.

- Traditional approach:
  - Indication of intermediate points between initial and final positions.
  - Choice of a shape of trajectory between the intermediate points in a repertory of forms provided by the control system (uncoordinated joint motions, straight lines in the space of the coordinates of the joints, straight lines in the Cartesian space).
  - Each form represents a different compromise between speed of execution and natural behavior.

Motion specification (cont'd)

- Traditional approach :
  - Non-uniqueness of correspondence « Cartesian coordinates » – « joints coordinates ».
  - Necessity of mechanisms, provided by the system, for the choice among alternative solutions (the VAL language provides for example a set of commands allowing the user to choose 1 from a maximum of 7 solutions of joint coordinates at certain Cartesian positions).
  - Unsatisfactory approach for robots with infinite families of solutions or for the specification of behavior at a kinematic singularity (for example, very fine control of velocity and shape of trajectory required in the case of spray painting).
  - Solution: provision of explicit trajectory control commands (parameterized procedures, etc.).

Control flow

- Need to be able to choose between several alternative actions for sensor-based robots.
- Detection and error correction (need for IF-THEN).
- Parallel executions.
- Need of mechanisms for the cooperation of robots (for example two robots that have to lift a load).
- Taking into account real time constraints .

Programming support

- Specificities of robotic programs:
  - Frequent access to external data.
  - Frequent Solicitation of users for corrective data or actions.
  - Production of statistical reports.
- Characteristics deserving special treatment :
  - Complex side effects and usually long run times
    - ⇒ Re-initialization difficult in case of failure.
    - ⇒ Need to allow online modification and restart.
  - Non repetitiveness of sensory information and of real-time interactions
    - => Need to record the traces of sensors same time as the program for debugging purposes.
  - Difficulty of visualization of geometry and complex motions
    - => Major role of simulators for debugging.

Robots programming approaches

- 3 approaches :
  - Guiding : the user guides the robot through the motion to perform.
  - Robot level programming : the user writes the computer program specifying motion and sensing.
  - Task level programming : the user specifies the operations by the effect he wishes they have on the objects.

Guiding

- The oldest approach and the most spread.
- Manual displacement of the robot to each desired position and storing coordinates of the joints for this position: operations such as object grasping possible at the position reached.
- Program : sequence of vectors of coordinates of joints + activation signals for the external equipments.
- Execution by displacement of the robot to the specified positions and generation of indicated signals.
- Method known as « teach by demonstration ».

Guiding: advantages and drawbacks

- Simple to use and to implement.
- Some limitations, in particular with respect to the use of sensors : the programmer specifies a unique sequence of execution for the robot (no loop, conditional, nor iteration instruction).
- Adapted for applications such as painting and management of simple materials.
- Not adapted for applications such as mechanical assembling or inspection where one needs to realize an action depending on the values obtained from sensors or from some data or from results of computations.

Robot level programming

- "robot level" programming languages incorporate commands to access sensor values and specify robot motions.
- Advantages :
  - Allow the use of data from external sensors (vision, force, etc.) to change the robot's motions.
  - Extends the scope of application of robots: they can face a greater degree of uncertainty..
- Main problem : require some expertise in computer programming and design of sensor-based motion strategies
  - ⇒ Inaccessible to the typical worker of the factory.
  - ⇒ Different works were carried out to extend the basic philosophy of guiding to include decision-making based on the value of the sensors.

Task level programming

- Specification of the objectives for the positions of the objects rather than the movements of the robot necessary to achieve these objectives.
- A specification at the task level is particularly supposed to be independent of the robot.
- Requires complete geometric models of the environment and the robot as input.
- Referenced as systems based on models of the world.

Categories of robotics programming languages

- 3 design approaches:
  - Extension of general purpose languages.
  - Creation of domain specific languages.
  - Modification of control languages.

Extension of general purpose languages

- Widespread use.
- Requires a certain power of expression.
- Creation of libraries dedicated to robotics.
- Advantages :
  - Possibility to use the same language at the different levels of the application.
  - Reduction of development load.
- Drawbacks :
  - Not adequate on the point of view of ease of specification and determinism of execution for complex robotic applications.

Example of extension : RCCL

- Collection of robot control routines provided as a C library.
- Has been successful since its delivery in 1983: Use by Jet Propulsion Laboratories, RCA Advanced Technology Labs and NASA Goddard Space Flight Center: Demonstration of the tasks required to maintain an Orbital Space Station.
- Offers facilities for:
  - The management of useful types such as 3D coordinate vectors, functions for coordinate transformation and coordinates system change.
  - Specify motions in the operating space or into the articular space of the robot.
  - Parameter motions (speed or execution time) and trajectory generator.
  - Synchronize the movements of several manipulators.

Example of extension (cont'd)

- ESL: extension of LISP, proposed by E. Gatt, mainly concerning :
  - task management with different synchronization possibilities.
  - Procedures for error recovery.
  - Goal management and how to achieve them.
  - The management of shared resources .
  - A small logical database (management of a global basis of assertions).
- LOGO:
  - Lisp very modified.
  - Has been used by MIT's AI laboratory and its Research-Education Unit.
  - Possibility of interfacing between Lego kits and LOGO on PC.

Creation of domain specific languages

- Domain specific languages : dedicated to programming of a field or a particular problem.
- Multiple interests :
  - Are closer to a specification language: while remaining executable, they hide the details of implementation.
  - Capture better the semantics of their domain and thus produce clearer and more concise programs.
  - Their semantic is often clear due to their limited expressivity.
- Drawbacks:
  - Are less expressive than general purpose languages and do not allow manipulation of complex data structures.
  - Low extensibility: difficult to adapt to a larger application domain.

Domain specific language examples

- KAREL:
  - Simple robotic programming language, initially intended for the teaching of classical computer programming.
  - Used by many students since 1981.
  - Structured in blocks as PASCAL language.
  - Supplies: position variables, speed control, motion control, input detection, output control.
  - First version used for simulated robots, but version of 1995 used for real robots in the laboratory. .
- VAL-2:
  - Extension of KAREL.
  - Interesting supply : fine motion production.
  - Possibility for the input sensors, to modify the motion of the controlled component (interest for example for clamping operations).
- Others : AML (A Manufacturing Language), …

Modification of control languages

- Control languages have an expressiveness closer to mission programming because they have been designed for parallel and responsive programming.
- Examples of control languages :
  - Synchronous languages data flows : LUSTER, SIGNAL, ...
  - Synchronous graphic languages: ARGOS, STATECHARTS, SYNCCHARTS, GRAFCET, ...
  - Textual control languages: ESTEREL.
  - Non-graphic asynchronous languages: ELECTRE, ...
  - Graphical asynchronous languages: Petri nets, etc. .
- Advantage: rigorously defined semantics and availability of simulation and / or verification and / or analysis tools.
- Drawback : requires greater expertise.

Examples of modification of control languages

- ESTEREL execution machine implemented in ORCCAD applications: Formalization, in the form of signals, of the dialogue between the controlled process (robotic actions) and the reactive controller.

  => Illustrates the need for adaptation.

Criteria for an ideal missions programming language

- Applicability of formal verification and analysis methods.
- Good expressiveness : basic control structures + structures for parallel and reactive programming.
- Intuitive programming of the control of applications, for example, graphical formalisms.
- Programming at robot task level.
- Good compromise between extensibility and specialization.
- Integration of operator intervention capabilities.
- Possibility to modify programs online.

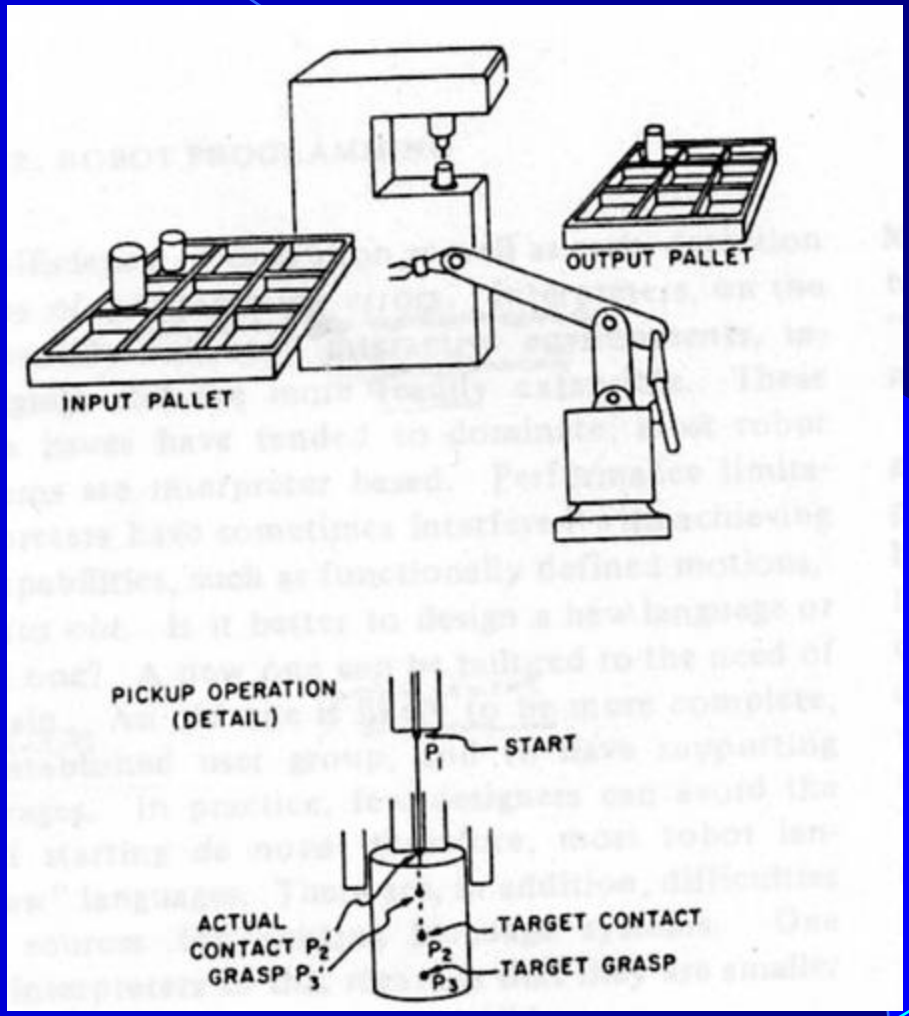
Investigation of some robot programming systems

- Guiding :
  - Extended guiding.
  - Offline guiding.
- Robot level programming.
- Task level programming.

Extended guiding

- Guiding system of ASEA robot:

• Taking simple forms of detection into account (guarded motion, ...).
• Provision of simple control structures, for example transfer of control at different points in the sequence taught.
• ASEA supports conditional branching and simple forms of procedures.
• The programmer can exploit these opportunities to produce more compact programs.



Program of the task

```
10   OUTPUT ON 17    Flag « ON » => take
20   PATTERN         Beginning of  procedure
30   TEST JUMP 17    Ignore this instruction  if « ON »
40   JUMP 170
50   OUTPUT OFF 17   Next time  put down
60   …               Operation « take»
100  MOD             End of code of  operation « take »
110  …               Positioning for  first pick up
130  MOD             Execute the procedure
140  …               Positioning for  second pick up
160  MOD             Execute the procedure
170  …               Machining and  deposit
200  OUTPUT ON 17    Next time « take»
210  MOD             End of code of operation « deposit »
220  …               Positioning for first deposit
230  MOD             Execute procedure
240  …               Positioning for second deposit
```

Offline guiding

- Use of a model of the task and a model of the robot.
- Simulation of robot movements in response to a program or guide input from a “guided teaching”.
- Advantage: flexibility and safety + possibility to experiment different configurations to find the most suitable (for example, minimal execution time).

Robot level programming

- MHI (Mechanical Hand Interpreter) is the first "robot level" programming language.
- Designed for one of the first computer-controlled robots, the MH-1 at MIT
- Programming style centered around motions with guard.
- Primitives of the language:
  - Move: indicates a direction and a speed
  - Until: tests a sensor against a specified condition.
  - Ifgoto: jump to a label if a certain condition is detected.
  - ifcontinue: branch to the continuation of the action if a certain condition is satisfied. .
- Did not support arithmetic control structures or other control structures beyond sensor control.

The PILOT Research project

- Context :
  - Craft programming of robots.
  - High development cost, without any warranty of missions success.
- Problematic: make the programming of robots or intelligent machines (autonomous or teleoperated) easier and more secure.
- Different works within our laboratory (Lab-STICC – UMR CNRS 6285, formerly LIMI/EA3883):
  - *YALTA : Yet Another Language for Telerobotics Applications* (thèse de J-C. Paoletti, 1991).
  - *PILOT: un langage pour la télérobotique* (thèse E. Le Rest, 1996).
  - *Vers une méthodologie de programmation d'un système de télérobotique : comparaison des approches PILOT et Grafcet* (thèse J.L. Fleureau, 1998).
  - Safety aspects of applications.
  - Verification of temporal properties : application of the results of the project « Distributed real time scheduling ».
- Test platforms: VESA II, MARC'H (BMO grant).

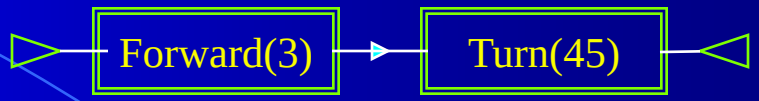
PILOT language

- *Graphical, interpreted* language, based on *the notion of action.*
- An *action* is an *object* that comprises :
  - A *name,*
  - A *type,*
  - A *precondition,*
  - *Supervising rules.*
- Control structures :
  - Classical : *sequentiality, conditional, iteration.*
  - Parallel programming : *parallelism, preemption.*

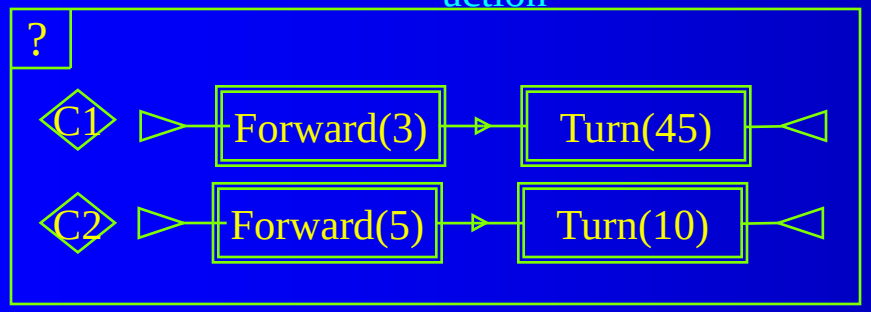
Actions and operators of PILOT language

Elementary action

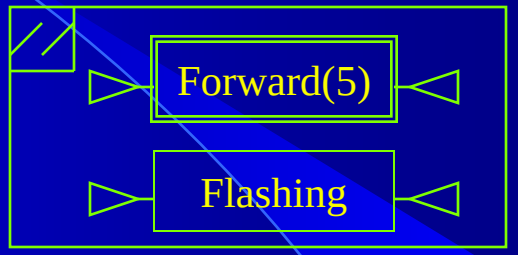
Continuous action



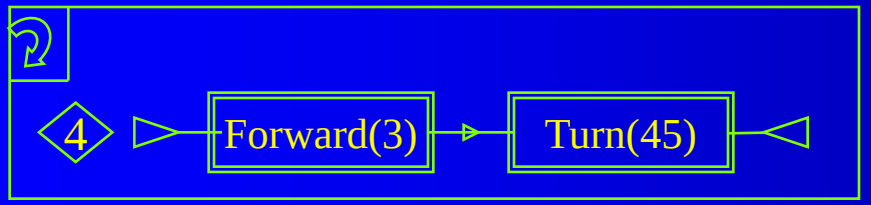
Sequentiality



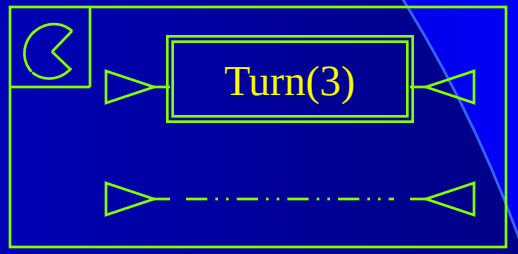
Conditional



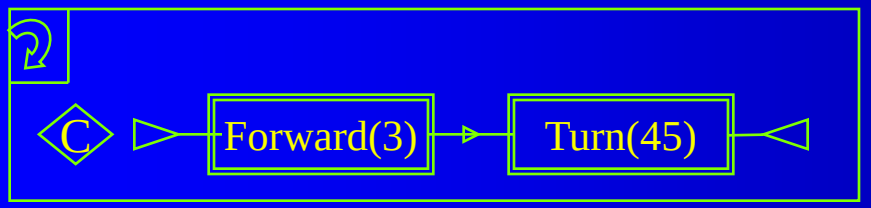
Parallelism



Fix iteration

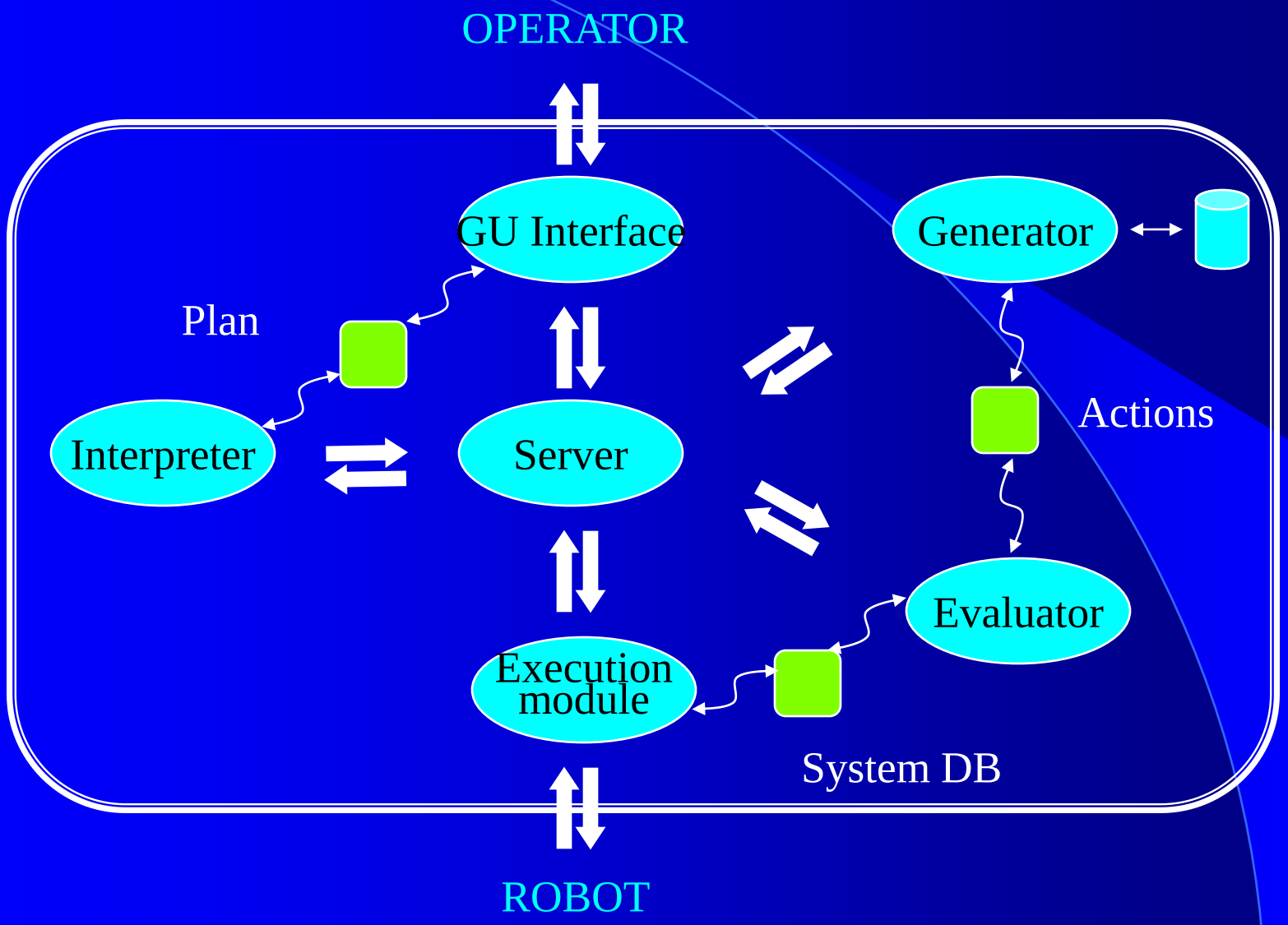


Preemption



Conditional iteration

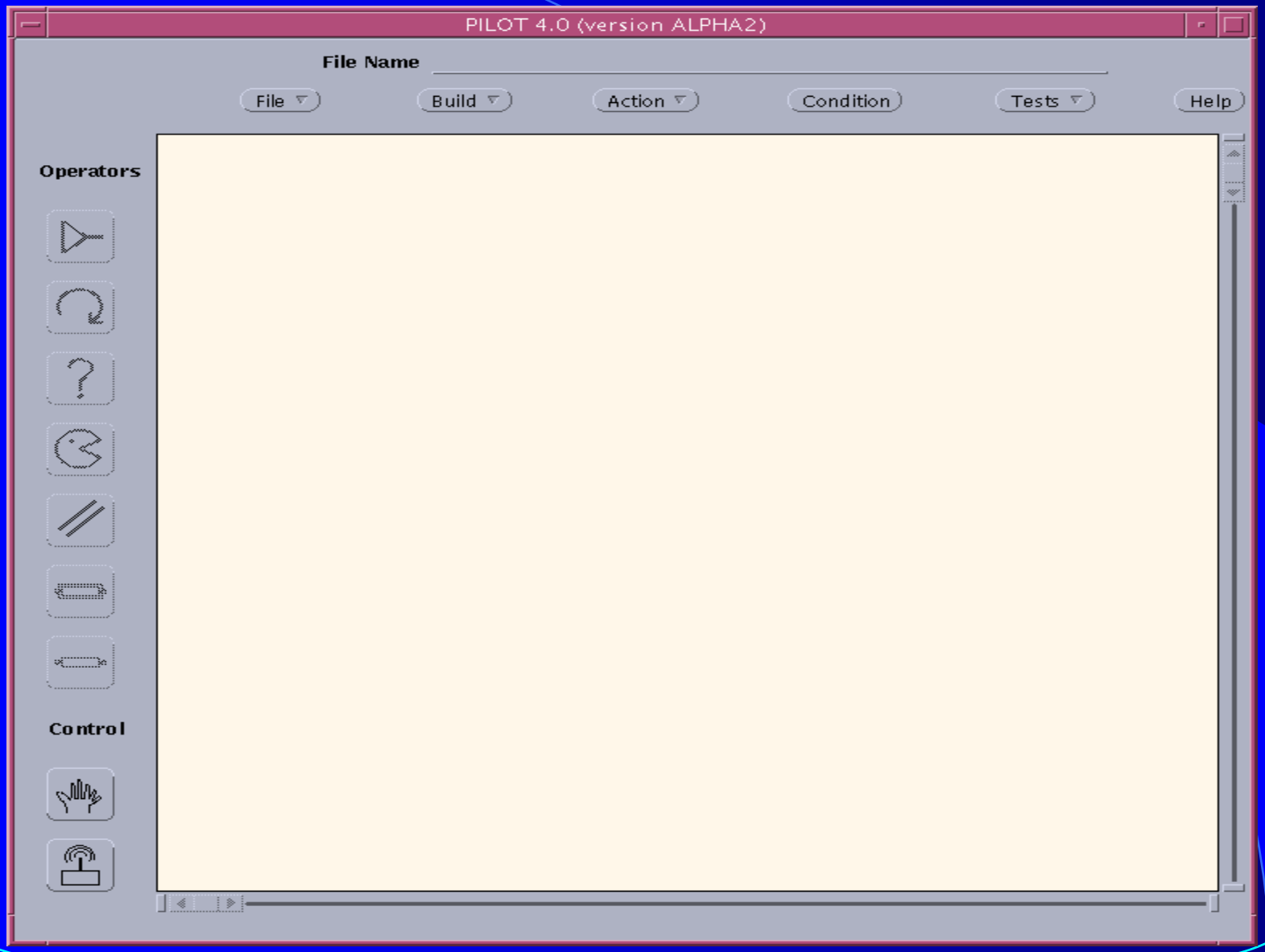
Software architecture of PILOT



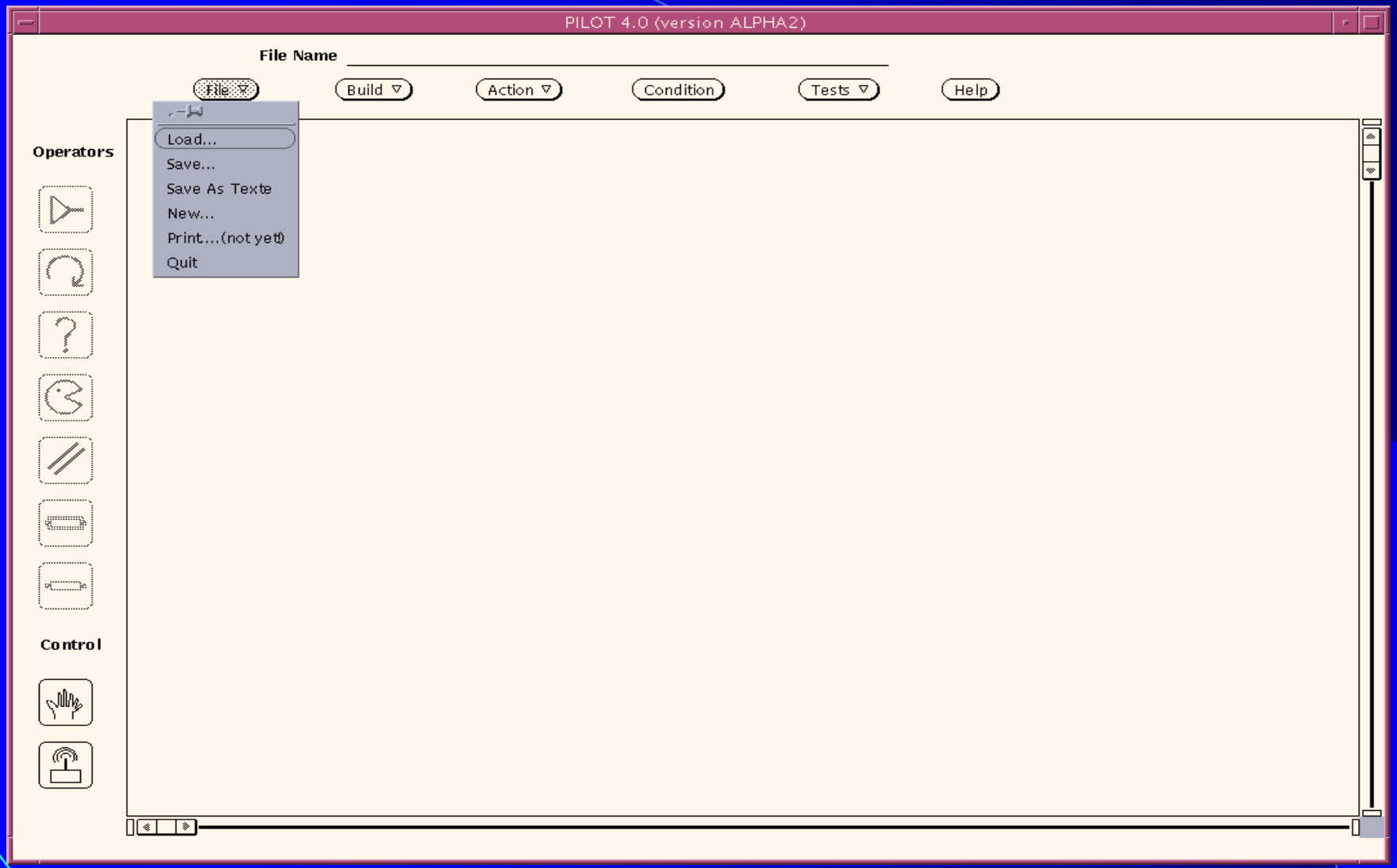
Functioning on user side

- Main window
- Creation and modification of actions
- Edition of plans
- Execution of plans

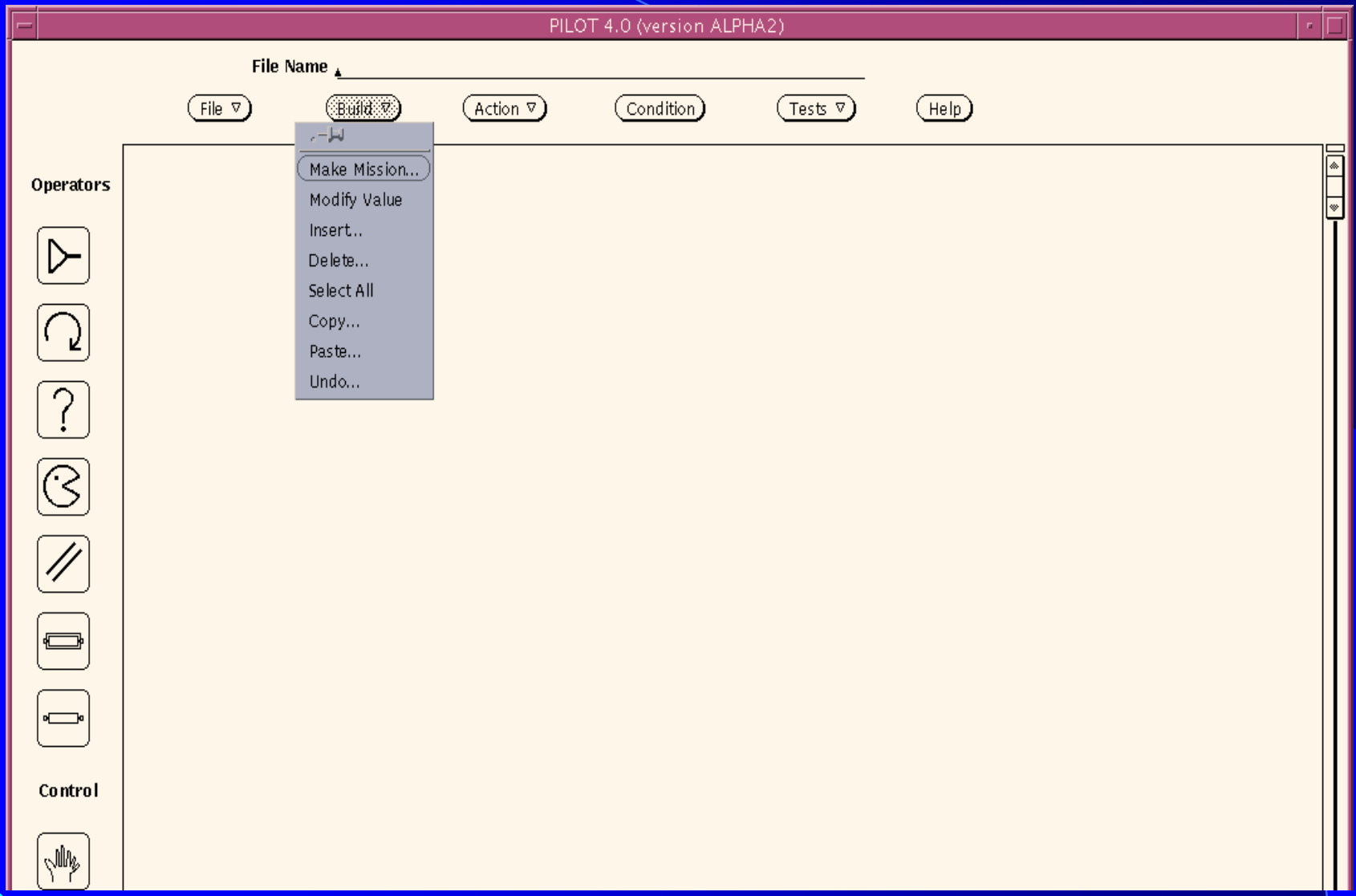
Main window


PILOT 4.0 (version ALPHA2)
File Name
File
Build
Action
Condition
Tests
Help
Operators
Control

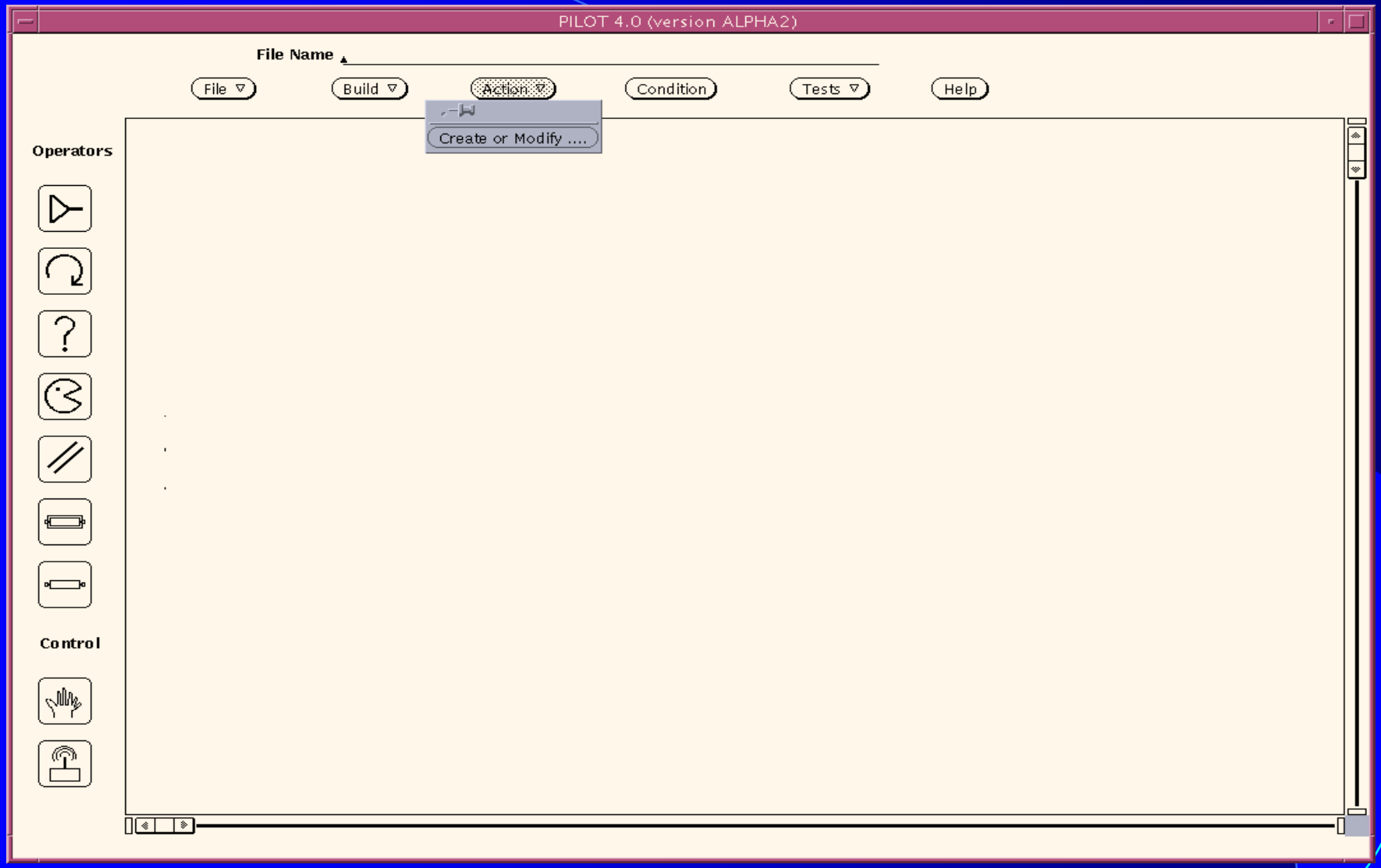
File Menu


PILOT 4.0 (version ALPHA2)
File Name
File
Build
Action
Condition
Tests
Help
Load...
Save...
Save As Texte
New...
Print...(not yet)
Quit
Operators
Control

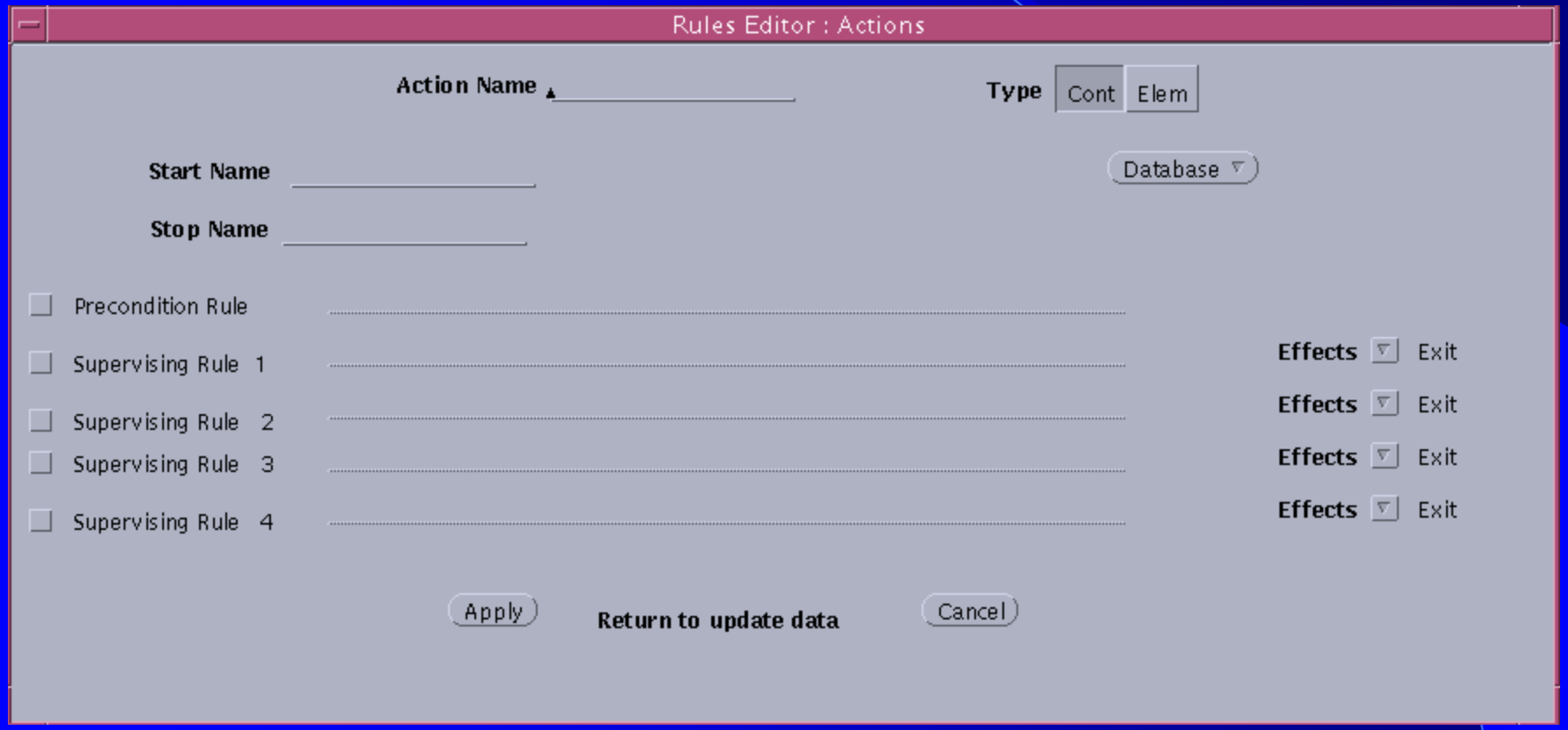
Build Menu


PILOT 4.0 (version ALPHA2)
File Name
File
Build
Action
Condition
Tests
Help
Make Mission...
Modify Value
Insert...
Delete...
Select All
Copy...
Paste...
Undo...
Operators
Control

Action Menu

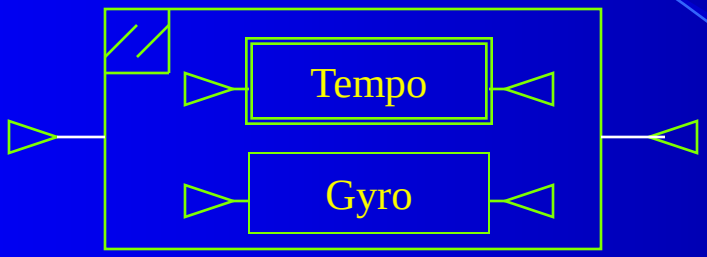

PILOT 4.0 (version ALPHA2)
File Name
File
Build
Action
Condition
Tests
Help
Create or Modify ....
Operators
Control

Creation and modification of actions



End-to-end behavior of Pilot control system

- Plan used for illustration



- Assumptions :
  - Plan execution has already been launched.
  - The plan is not modified during its execution.
  - Preconditions :
    - Tempo : NOT *timer_on*
    - Gyro: NOT *ligth_on*
  - Supervising rules (only one for each action):
    - Tempo : NOT *timer_on*
    - Gyro: NOT *light_on*

Description of the behavior

- HMI : send message Start_Execution to interpreter.
- Interpreter :
  - Starts interprétation upon reception.
  - Interpretation of the beginning node of the sequence : boils down to the interpretation of the following node => interpretation of the parallel structure.
  - Allocation of a END_BOX structure
  - Storage of the start and end of sequence pointers of the parallel structure in the latter.
    - Sequential analysis of the first elements following the beginnings of internal sequences, ie tempo then gyro analysis.
    - Allocation of a *cell* for tempo and positioning of its fields (locked , Elementary Action type, precondition request state, synchronization, pointer on END_BOX, ...).
    - Similar allocation for Gyro.
- Interpreter : sending requests for evaluation of pre-conditions of *tempo* and *gyro* to the evaluator and waiting on message queue.
- HMI : update of the status of the actions.

Description (cont'd)

- Evaluator :
  - After receiving the request, reads and evaluates the preconditions of tempo and gyro.
  - Sends message PRECONDITION_ACCEPTED (assuming that pre conditions are satisfied) to the interpreter for each action.
- Interpreter :
  - After receiving the message, sends message START_ACTION to both EVALUATOR and EXECUTION modules.
  - Positions the state of actions to ACTION_LAUNCHED.
- HMI : graphic update of actions' state.

Description (cont'd)

- Execution : sends orders of execution of *tempo* and *gyro* to the robot.
- Evaluator : Positions the state of actions *tempo* and *gyro* to *Ongoing* and starts evaluation of monitoring rules.
- Robot :
  - Triggers the « timer » for the specified duration and turns on the flashing light.
  - When the duration expires, sends information « tempo action completed » to the execution module.
- Execution:
  - Gets sensors information from the robot, including those related to the timer and the flashing light (timer_on, gyro_on) and stores it into shared memory.
  - Same for information on the action tempo.

Description (cont'd)

- Evaluator:
  - Detects the end of the action « *tempo* » and positions its state to « completed »,
  - Stops the evaluation of corresponding monitoring rules.
  - Sends message ACTION_COMPLETED to the interpreter.
- Interpreter :
  - Positions action « *tempo* » to COMPLETED => graphical update by the HMI.
  - Starts interpretation of the next primitive i.e « end of sequence » => call of the termination detection algorithm that looks for the direct primitive englobing the end of sequence, i.e the parallel box, and examines the corresponding END_BOX in order to verify if the condition of end of parallel execution is satisfied.
  - End of parallel execution is detected => sends a message STOP_ACTION to the evaluator and to the execution module for the action « gyro », and positions the state of the action « gyro » to COMPLETED => graphical update is done by the HMI.

Description (cont'd and end)

- Evaluator : positions the action « *gyro* » to STOPPED and stops the evaluation of its monitoring rules.
- Execution : sends stop order for the action « gyro » to the robot which stops the flashing light.
- Interpreter :
  - Analyses the node that follows the parallel box, i.e the end of the main sequence => the termination detection algorithm is called again and detects the end of the mission.
  - Sends the message MISSION_COMPLETED to the other modules and all these modules prepare themselves for the execution of the next mission or for the end of the application.