Robotica Mobile

Lezione 7:
Uno sguardo dentro Saphira

5-05-2004

La struttura del sis

- Parte sincrona
- Parte Asincrona

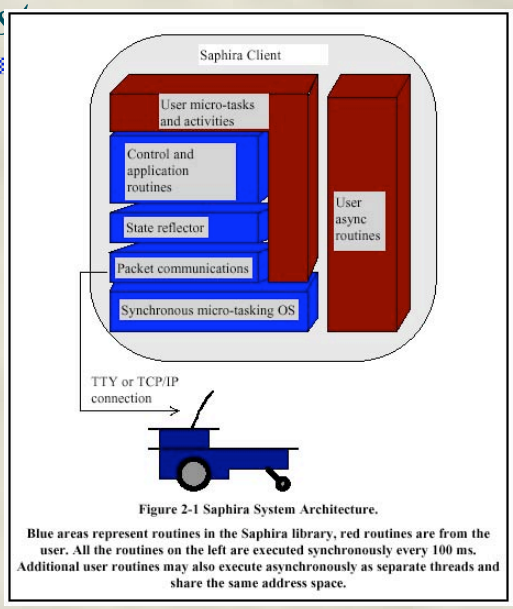


Figure 2-1 Saphira System Architecture.

Blue areas represent routines in the Saphira library, red routines are from the user. All the routines on the left are executed synchronously every 100 ms. Additional user routines may also execute asynchronously as separate threads and share the same address space.

Lezione 7: Uno sguardo dentro Saphira 5-05-2004

Architettura di controllo:

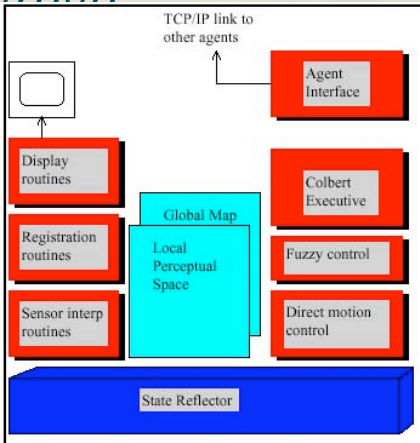


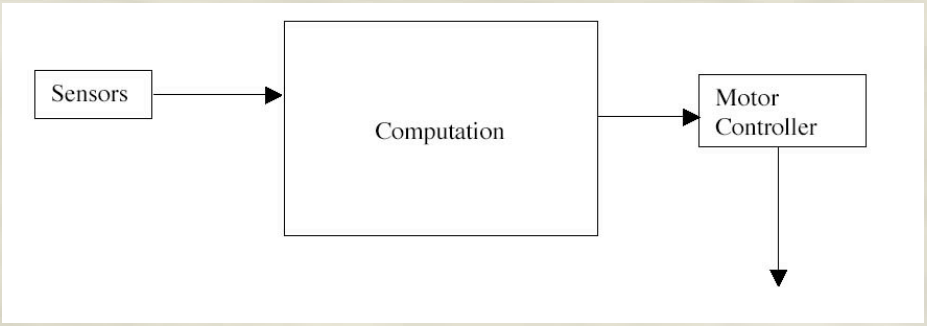
Figure 2-1. Saphira's Control Architecture.

The control architecture is a set of routines that interpret sensor readings relative to a geometric world model, and a set of action routines that map robot states to control actions. Registration routines link the robot's local sensor readings to its map of the world, and the Procedural Reasoning System sequences actions to achieve specific goals. The agent interface links the robot to other agents in the Open Agent Architecture.

Lezione 7: Uno sguardo dentro Saphira

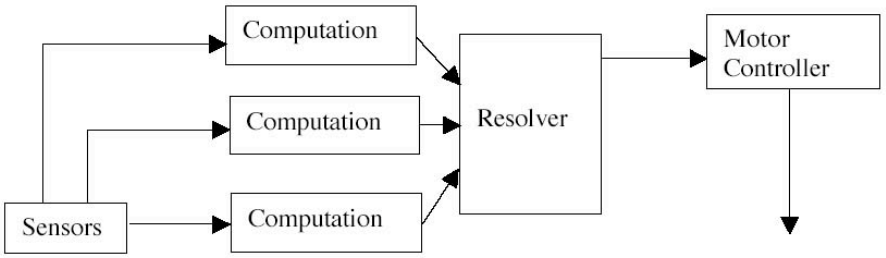
Le ragioni di questa complessità (1)

- Sistema di controllo tradizionale (monolitico):



Le ragioni di questa complessità (2)

- Controllo partizionato (indiretto):



Interazione Saphira-Pioneer

- L'host (reale o simulato) comunica le proprie caratteristiche a Saphira
- Quando si usa il simulatore, occorre ricordarsi di caricare i parametri giusti! (pion1m)

Un esempio di programma in Colbert

```
/* Colbert example exercising the direct motion calls */
act patrol(int a)               /* go back and forth 'a' times */
{
while (a != 0)
{
turnto(180);
     a = a-1;
move(1000);
turnto(0);
move(1000);
}
}

act square                      /* move in a square */
{
int a;
a = 4;
while(a)
{
a = a-1;
move(1000);
turn(90);
}
}

act aa                          /* call them sequentially */
{
trace patrol;
start patrol(4);
trace square;
start square;
}

sfSetDisplayState(sfGLOBAL, 1); /* put display into global coords */
start aa;                       /* start up the toplevel activity */
```


Fondamentali comandi di movimento diretto

Command	Effect
`move(int mm);`	Move the robot `mm` millimeters forward (positive) or backwards (negative). Blocking.
`turn(int deg);`	Turn the robot `deg` degrees clockwise (negative) or counter-clockwise (positive) degrees from the current heading. Blocking.
`turnto(int deg);`	Turn the robot to the heading `deg` degrees. Positive values are counter-clockwise, negative values are clockwise. Blocking.
`speed(int mms);`	Move the robot at a speed of `mms` millimeters per second forward (positive) or backwards (negative). Non-blocking.
`rotate(int degs);`	Move the robot at a rotational speed of `degs` degrees per second counter-clockwise (positive) or clockwise (negative). Non-blocking.
`halt;`	Halts all direct motion commands.

La documentazione di SAPHIRA-ARIA

- Tutta sul sito!

E adesso passiamo alle prove pratiche!

Finora…

- Abbiamo visto alcune caratteristiche essenziali di Colbert
- Riprendiamo i punti più importanti
- Le funzioni a disposizione sono pochissime!
- Perché?

Gli stati delle activity

- `suspended` If an activity is suspended, it does not contribute to the behavior of the robot, and the Colbert executive skips processing it. A suspended activity can be restarted using the `resume` command or the `start` command.
- `timedout` The timed out state is similar to the suspended state, in that no execution takes place. It is the result of the activity using up its allotted time, given on invocation with the `timeout` parameter.
- `success` This state is used by an activity to indicate that it has successfully completed its processing. No further execution takes place, unless the activity is signaled with a resumption signal.
- `failure` Similar to success, but the activity has unsuccessfully completed its processing.

Gli stati di una activity, e come acquisirli

State	Numeric value	Symbolic value
suspended	1	SfSUSPEND
timedout	5	SfTIMEOUT
success	3	SfSUCCESS
failure	4	sfFAILURE

-
```
> sfGetTaskState("bng");
Eval to (int) 12
> sfGetTaskState("Wander");
Eval to (int) 1
```
- `sfTaskFinished(<instance_name>)`
- `sfTaskSuspended(<instance_name>)`

An activity is finished if its state is one of the completed states: timed out, success, or failure. If the named activity instance does not exist, sfTaskFinished will return true (1).

An activity is suspended if it is in either the suspended or interrupted state. If the named activity instance does not exist, sfTaskSuspended returns false (0).

Signals

Signal	Colbert command	C++ command
remove	remove iname	sfRemoveTask(char *iname)
suspend	suspend iname [n]	sfSuspendTask(char *iname, int time)
interrupt	interrupt iname	sfInterruptTask(char *iname)
resume	resume iname	sfResumeTask(char *iname)
success	succeed iname	sfSucceedTask(char *iname)
failure	fail iname	sfFailTask(char *iname)

```
act bng()
{
  while(1)
  {
    waitfor(sfStalledMotor(sfLEFT) || sfStalledMotor(sfRIGHT));
    remove getout;                  // halt this activity if it's going
    stop();                         // stop the robot
    move (-BACKVAL) timeout 30;     // just in case, we timeout here
    start getout(MOVEVAL, TURNDEG) noblock;
  }
}
```