URBI – Universal Robotic Body Interface

Jean-Christophe Baillie

Ecole Nationale Supérieure des Techniques Avancées
Paris, France

Outline

• Introduction
• Devices
• Examples
• Soft devices
• URBI usages
• Conclusion

URBI is an Interface Language to control robots

Key features:

1. Simplicity: easy to understand, but with high level capabilities
2. Flexibility: independent of the robot, OS, platform, interfaced with many languages (C++, Java, Matlab…)
3. Modularity: soft devices to extend the language
4. Asynchronicity: Parallel processing of commands, concurrent variable access policies, event-based programming,…
Every sensor, motor, camera or physical hardware in the robot is a **device**.

A device is similar to a C++ object: it has **methods** and **properties**. The **val** property is related to the device **value**.

```
// The ball tracking program:
whenever (ball.visible) {
    headPan.val = headPan.val + camera.xfov * ball.x
    headTilt.val = headTilt.val + camera.yfov * ball.y
};
```

```
// Stand up (leg1, leg2 and leg3 are virtual devices defined elsewhere)
getup: |
{ leg2.val = 90 time:2000 & leg3.val = 0 time:2000 } |
leg1.val = 90 time:1000 |
leg2.val = 10 time:1000 |
{ leg1.val = -10 time:2000 & leg3.val = 90 time:2000 }
```

For more details on **for/while/if**, **functions**, **events**, **variables** & more, visit [www.urbiforge.com](http://www.urbiforge.com).
**URBI – Universal Robotic Body Interface**

- **Making reusable software modules**

  **Algorithm results as URBI variables**
  
  Ex: ball.x, ball.y, ball.size

  **External client module**

  Compute some complex function, algorithm or signal processing task.
  
  Example: a ball tracking algorithm.

  **Activity script**:

  ```
  loop modulename:camera.val;
  Controlled by: block modulename;
  stop modulename;
  ```

  **Sensors or variable values**
  
  Ex: camera.val

  **Client program or URBI script**

  Make use of the URBI variables constantly updated by the client module.

---

**URBI usages**

- **Possible standard?**
  
  For robotics and smart objects in general

- ✓ Boost robotic software industry
- ✓ Enable home shared computing
- ✓ Educational tool

**Future extensions: Server = Client**

(client functions included in the language)

**Query methods**:

```
devices; // return the list of devices
info headPan; // info about a device
```

- ➔ Each component can ask about what are the available components around and what they can do. Then, it can start to interact with it.

---

Jean-Christophe Baillie
URBI Server for Aibo, Webots & liburbi are freely available for non commercial use at:

www.urbiforge.com

Next steps:

• Support **more robots** and **develop partnership/contracts with device manufacturers**:
  
  Currently: Aibo, HRP-2, Webots5 (robot simulator, Cyberbotics), Aldebaran Robotics
  
  Next: Sony, ActivMedia, Philips & iCat?

• **Increase the number of languages** for the liburbi:
  
  Currently: C++, Java, Matlab, C++/OPENR.
  
  Next: Python, C, perl

• **Continue to add important features** in the kernel: client functions, debugging facilities, multi tagging, kernel 2 with multicore processors support and real-time scheduling.

• Develop **URBI plugins** to enhance the language and stimulate the community (GPL)

• Develop useful **associated software & tutorials**: URBI Lab, URBI center, URBI Dev

---

Jean-Christophe Baillie

---

Demo

Short demo video:
at (talk.finished == true)
  echo « Thank you for » +
  « your attention »;

Simple assignment:

```
headPan.val = -2;
```

Numerical assignments can be specified via `modifiers`

```
headPan.val = 15 time:5000;
headPan.val = 15 speed:0.34;
headPan.val = 15 accel:0.02;
headPan.val = -2 sin:1000 ampli:3;
```

Any function can be assigned as time parameterized trajectory with the function modifier (0.9.7):

```
headPan.val’n = function(t):sqr(t)+sin(3*t+pi);
```
Commands can be executed in **serial** or **parallel** mode:

- `headPan.val = 15 & neck.val = 30;` Set headPan to 15 and neck to 30 at the same time.
- `headPan.val = 15 | neck.val = 30;` Set headPan to 15 and after, set neck to 30.

Operators `&` and `|` are also available and have a semantics identical to `&` and `|` except that they have looser constraints:

- `A & B` (A and B execute at the same time)
- `A | B` (A before B)
- `A ; B` (A starts before B, A finishes before B)
- `A , B` (A finishes before B)

NB: Brackets can be used to group commands, like in C:

```
{ headPan.val = 15 | headTilt.val = 23 time:1000 } & neck.val = 10;
```

**Event catching**

Several event catching mechanisms are available:

- `at (test) { instructionsA; onleave { instructionsB; } };` A will be executed once when `test` becomes true. Then, as soon as `test` becomes false, B is executed and the server waits for `test` to be true again. `test` can be set with an *hysteresis* on the number of successful test checks or time.

- `whenever (test) { instructionsA; else { instructionsB; } };` When `test` becomes true, A is executed. When A is finished, it is executed again if `test` is still true, otherwise, B is executed.

- `waituntil (test);` Terminates only when `test` becomes true.

**In development:** `emit/catch events with parameters`
Variables in URBI are not simple containers like in C or similar languages:

they are **dynamic structures** evolving over time.
Devices can be grouped into virtual devices to form a hierarchy:

```
group legLF {lefLF1, legLF2, legLF3};
group legs {legLF, legLH, legRF, legRH};
```

Commands go down the hierarchy, allowing useful factorizations:

```
legs.PGain = 0; // will affect the P Gain of all sub devices

The @ prefix prevents the recursive descent:  @legs.PGain = 0;
```

Telnet is of course too limited => liburbi for C++ programming.
Exists also for JAVA, Matlab and OPENR (soon: Python, C)

The URBI library (liburbi) give simple methods to send commands to and receive messages from the URBI server.

C++ example:

```
UClient * client = new UClient("myrobot.ensta.fr");
client->send(" headPan.val = \t\t" , x);

client->syncGetDevice("neck",&neckVal); // exists also for binary

client->setCallback(&receiveMyImage,"imgtag");
client->send("loop imgtag:camera.val;");

UCallbackAction receiveMyImage(const Umessage &msg)
{
    /* handles the image contained in Umessage */
}
```
How to use URBI?

C++ client

```cpp
// C++ code with liburbi C++

UClient * client = new UClient("myrobot.ensta.fr");
int pos = complex_calculation(x,y);
client->send("neck. val = %d;", pos);
```

Java client

```java
// Java code with liburbi Java

import liburbi.UClient;
robotC = new UClient(robotname);
robotC.send("motoron ;");
robotC.setCallback(image, "cam");
```

Onboard client

```cpp
// C++ code with liburbi OPEN-R

UClient * client = new UClient("localhost");
int pos = complex_calculation(x,y);
client->send("neck. val = %d;", pos);
urbi::execute();
```

telnet or urbilab client

```bash
head feet val = thr
```

other integrated clients
(matlab, python, . . .)

Simple commands
- functions definition
- complex scripts

Soft Devices

The creation of "soft" devices extends the devices available in the robot. This is done by creating a subclass of USoftDevice. Soft devices can monitor variables, redirect function calls (C++/URBI binding) and events relative to them.

Example:

```
ball => ball.x, ball.y
voice => voice.say("Hello"), voice.hear(x)
```
Performances

Test on a wifi 802.11B wireless connections, with a URBI server for ERS7, a linux C++ client and tests with both the client and the server on the robot, and OPEN-R based connection.

- TCP/IP:
  - Latency: 1.5ms ± 0.5ms
  - Bandwidth: ~500Ko/s

- OPEN-R:
  - Latency: 600 µs
  - Bandwidth: ~2Mo/s

- Jpeg compress (90%):
  - 10ms for 208x160
  - 3ms for 104x80

Good performances in general. For highly demanding low level action/perception loops, the best is to run some URBI Clients on the robot and leave the high level slow control architecture offboard.

Jean-Christophe Baillie