THE OROCOS PROJECT

Open Robot Control Software

EURON Interest Group

Modular Architecture

Orocos @ LAAS

Anthony Mallet, Sara Fleury, Raja Chatila

LAAS - CNRS

mallet@laas.fr, sara@laas.fr, raja@laas.fr

Leuven, July 28th, 2003

Context and Objectives

Autonomous Systems

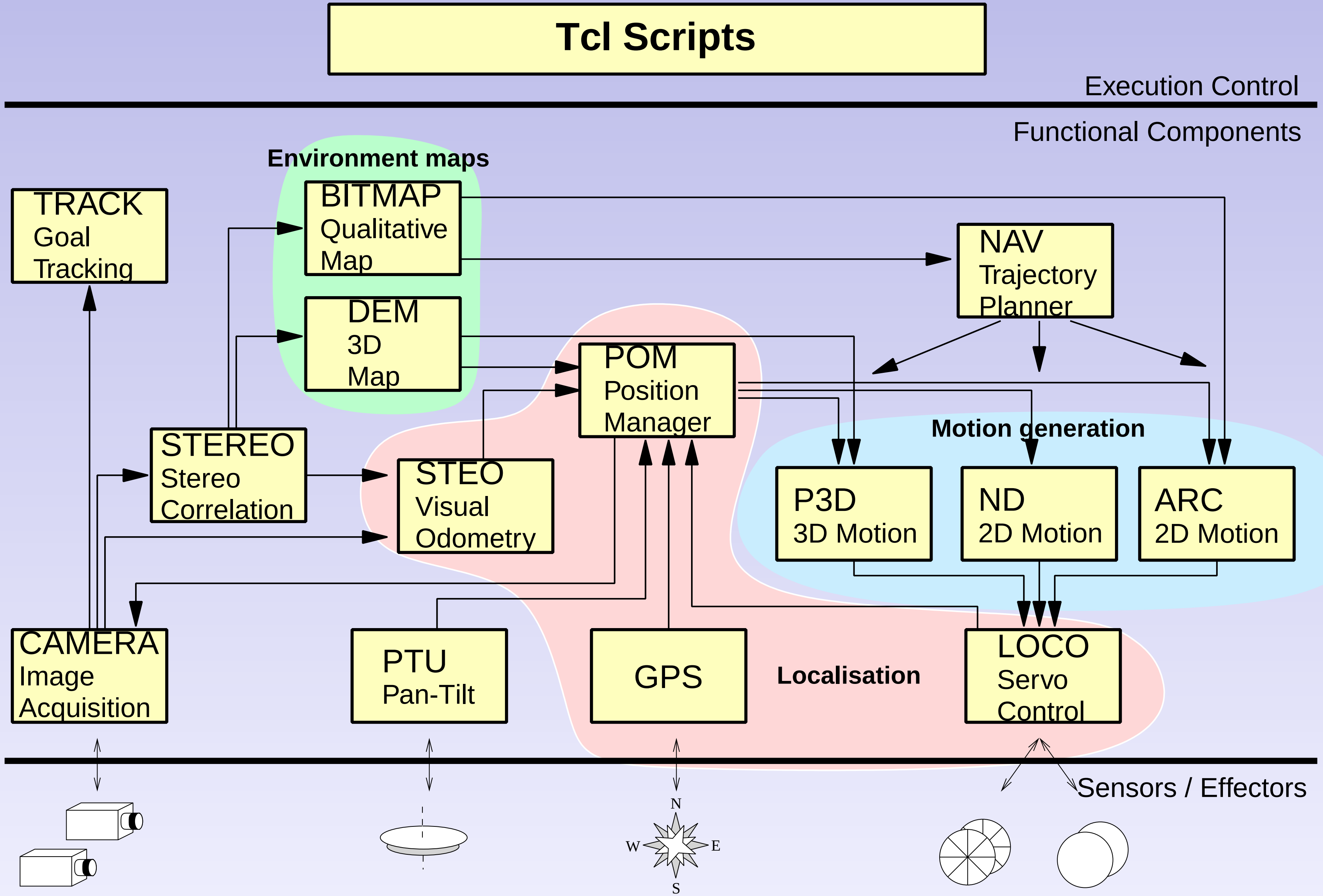
- Embedding perception, action and decision.
- High level commands (‘goto’, ‘explore’, ‘map’).
- Generic approaches - not focused on a particular application.

Objectives

- Development of functionalities
  - Device drivers or higher level, well known algorithms.
  - Repository of ’state of the art’ functions.
- Integration
  - Software architecture, development tools.
  - Real-time constraints, modularity, reusability.

Real example

Real example



Adherence to Workplan

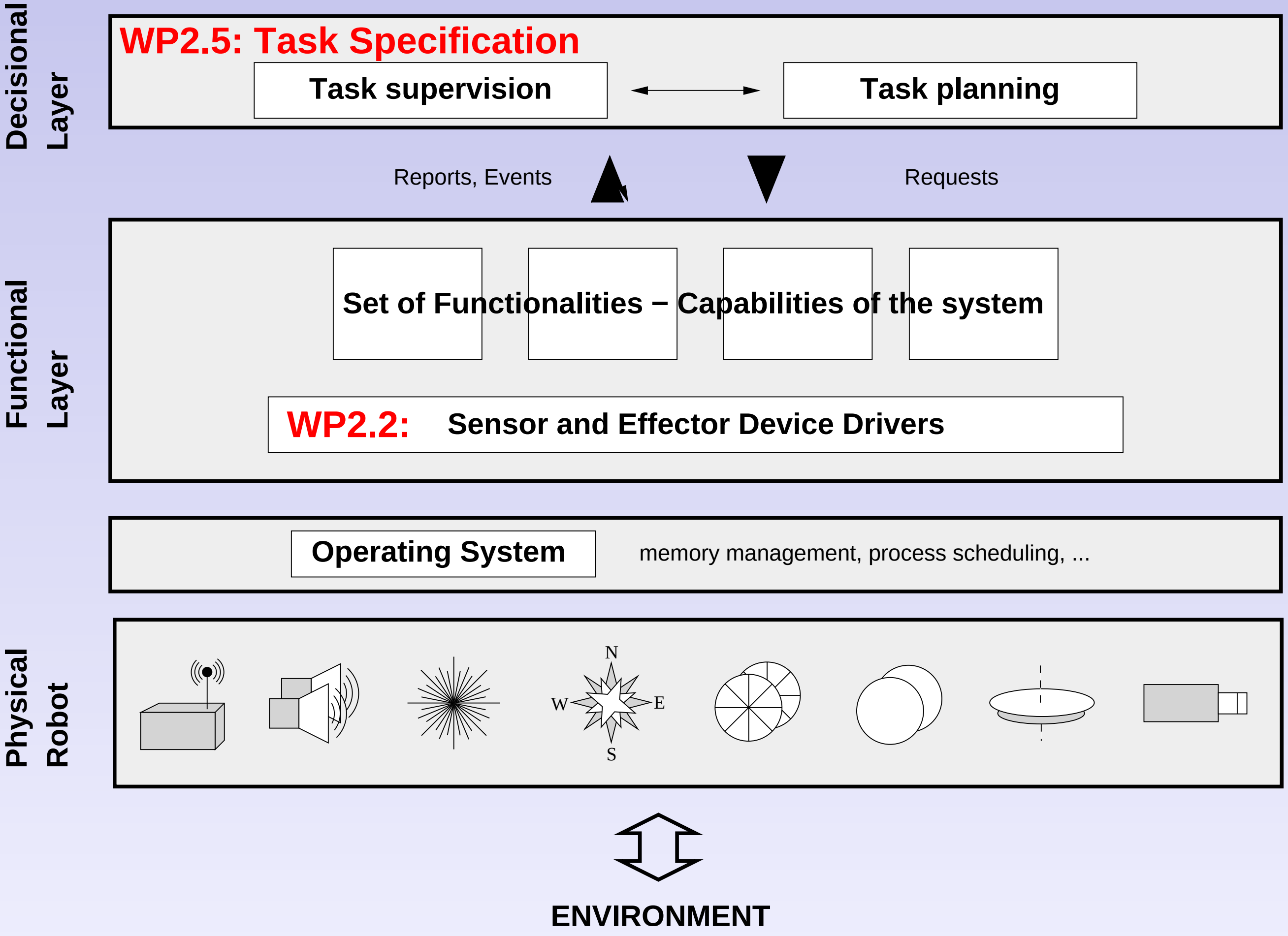
- **WP2.2: Sensor device drivers**
  - There is already a huge collection of software for most common robotics devices.
  - Incremental developpment, very heterogenous.
- **WP2.5: Task specification**
  - Needs a critical mass of functions.
  - The set of functions define the vocabulary of a task specification language.

→ **Fill the gap** between the two workpackages. → Definition of a **software architecture** and **integration tools**.

Presentation Outline

- **Software Architecture**
  - Separation between functions and decision.
  - Components definition and architecture.
- **GenoM: an instantiation of architectural concepts**
  - Principle, examples.
  - Evolutions.

- **Separation** between **functions** (capabilities) and **decision** (activation of capabilities) to deal with **changing, unpredictable or unknown environments**.
- Functional layer: **component-based** architecture.


Decisional Layer
WP2.5: Task Specification
Task supervision
Task planning
Reports, Events
Requests
Functional Layer
Set of Functionalities − Capabilities of the system
WP2.2: Sensor and Effector Device Drivers
Operating System
memory management, process scheduling, ...
Physical Robot
N
W
E
S
ENVIRONMENT

Components and Libraries

WP2.5 **Component:** A software component is a **binary, executable** object, should be **reusable**, be able to **address** other components and be addressed itself.

Components provide *services* to the decisional layer.

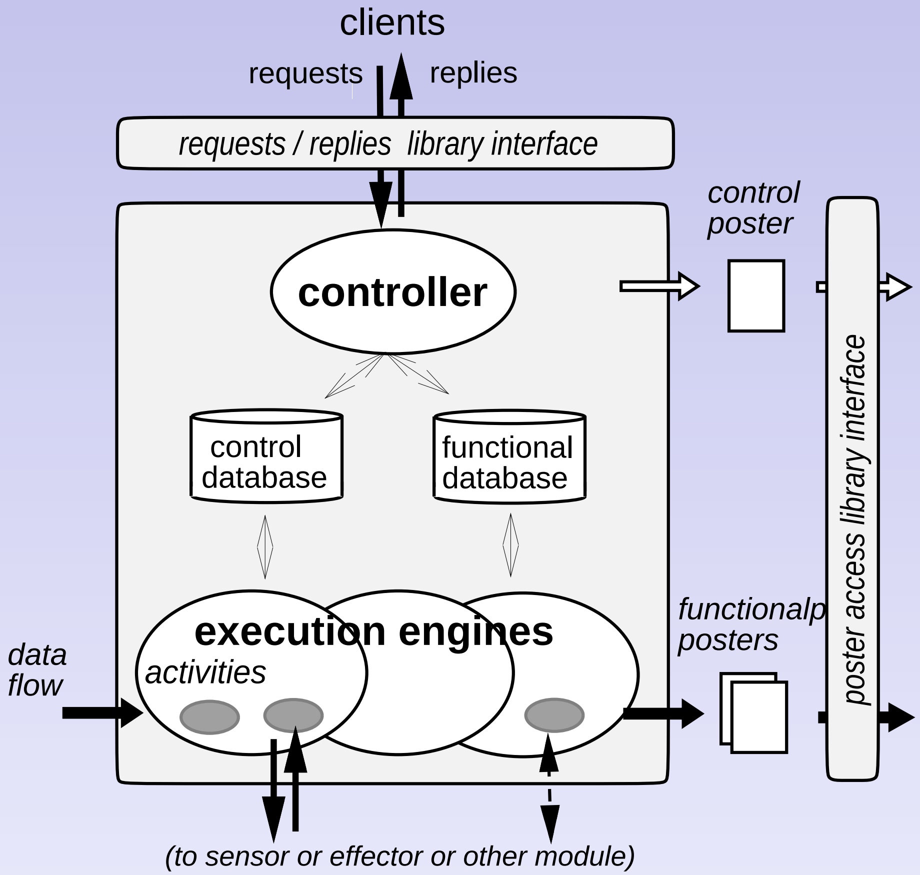
WP2.2 **Library:** Embed an implementation of one or several algorithms. This is a collection of code providing some *functionnality*.

Components represent the encapsulation of libraries in the context of autonomous systems.

- Libraries are reusable in other contexts.
- Components handle the real-time aspects and robotics constraints.
- Components are the basic building blocks for the task specification.

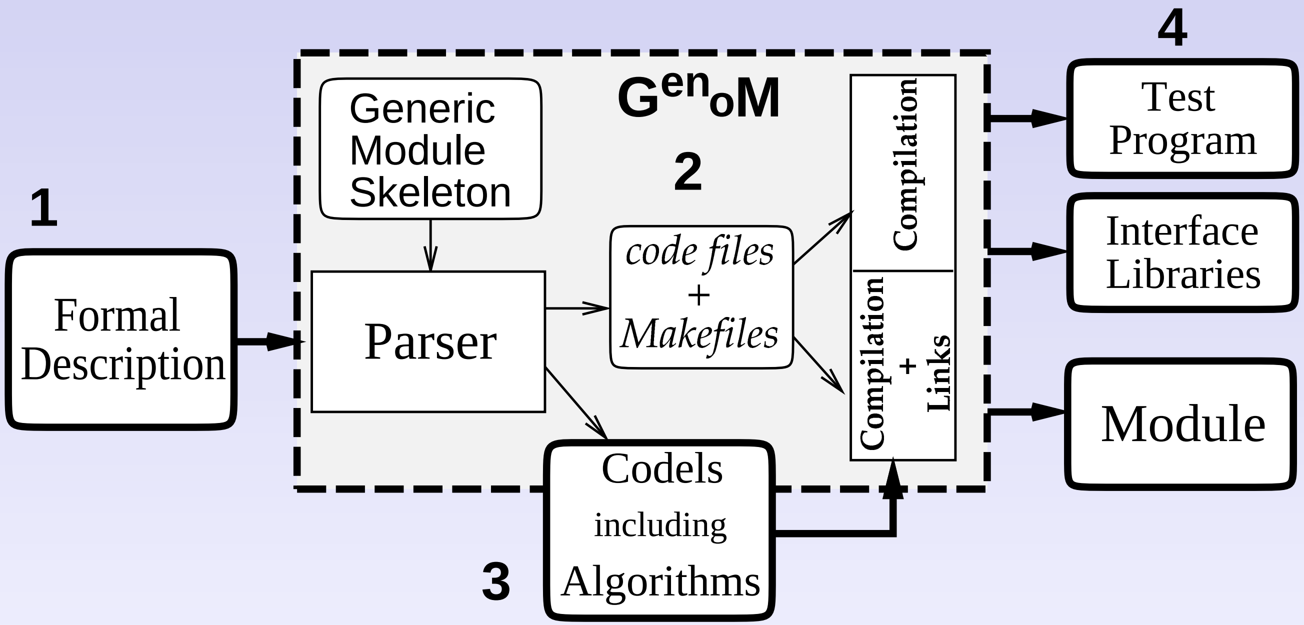
Starting point for OROCOS

- **Execution engine**: generic template.
- **Communication**: home-made real-time library (VxWorks and POSIX systems). Decoupling between control flow and data flow.
- **Libraries**: algorithms are structured in *codels*.
- **Codels**: are the smallest unit of code managed by the execution engine. Determine the *reactivity* of the component.



GenoM: component description + a set of libraries = one component.

- Formal description of the underlying functionnalities.
- Automatic generation of code (→ validation).
- “Developper friendly” (easy to learn, easy to use).



Example: Stereo-correlation

Libraries

- Image acquisition.
- Image processing (subsampling, distortion, ...).
- Image correlation and 3D reconstruction.

Components

- Image acquisition, with preprocessing capabilities.
- 3D image reconstruction.

Example: Components Description

Image Acquisition

```
#include "cameraStruct.h"

module camera {
    internal_data:          CameraIDS;
};

request OneShot {
    doc:                    "Acquire one (pair of) image(s)";

    input:                  imageParams::imageParams;
    input_info:
        CAMERA_BANK_A::"Which bank",
        CAMERA_STEREO::"Which image";

    c_exec_func:            cameraOneShot;

    exec_task:              ExecTask;

    fail_msg:               NOT_INITIALIZED, NOT_CONFIGURED, ...
};
```

Stereo Correlation

```
#include "scorrelStruct.h"

module scorrel {
    internal_data:          ScorrelIDS;
};

request SCorrel {
    doc: "Compute disparity and 3d image";

    c_exec_func_start:      scorrelStart;
    c_exec_func:            scorrelExec;
    c_exec_func_end:        scorrelEnd;
    c_exec_func_inter:      scorrelInter;

    exec_task:              ExecTask;

    fail_msg:               NO_CAMERA_POSTER, ...
};
```

Image Acquisition

```
ACTIVITY_EVENT
cameraOneShot(CameraBankImage *r, int *report)
{
   if (cameraAcqRead(image, width, height, report) != OK)
      return ETHER;

   if (cameraPreproc(image, width, height, report) != OK)
      return ETHER;

   if (posterTake(posterId, POSTER_WRITE) != OK) {
      camera_error("cameraOneShot", report, CANNOT_TAKE_POSTER);
      return ETHER;
   }

   cameraPosterImageCopy(poster, image, width, height);

   posterGive(posterId);

   return ETHER;
}
```

Example: Codels

Stereo Correlation

```
ACTIVITY_EVENT
scorrelSCorrelStart(int *report)
{
   if (cameraOneShotRequestSend( ... , report) != OK) {
      return ETHER;
   }

   return EXEC;
}

ACTIVITY_EVENT
scorrelSCorrelExec(int *report)
{
   if (cameraOneShotReplyReceive( ... , report) != FINAL_REPLY) {
      return EXEC;
   }

   posterTake(CAMERA_POSTER_ID, POSTER_READ);
   scorrelReadImage( ... );
   posterGive(CAMERA_POSTER_ID);

   return END;
}
```

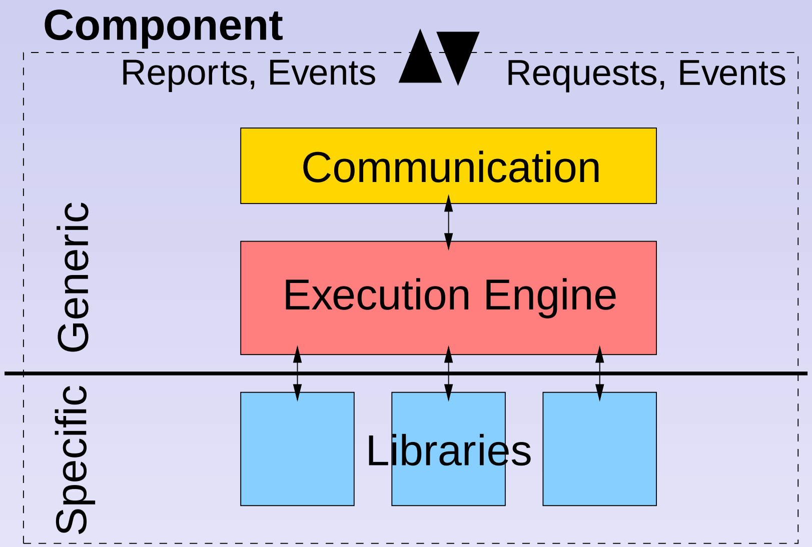
Stereo Correlation (2)

```
ACTIVITY_EVENT
scorrelSCorrelEnd(int *report)
{
   static int state;

   switch(state) {
      case 0:
         correlation_zncc( ... );
         state = 1;
         break;

      case 1:
         depth_image( ... );
         state = 2;
         break;

      case 2:
         reconstruction( ... );
         state = 0;
         break;
   }

   return (state == 0) ? ETHER : END;
}
```

Components Architecture

Requirements for functional components:

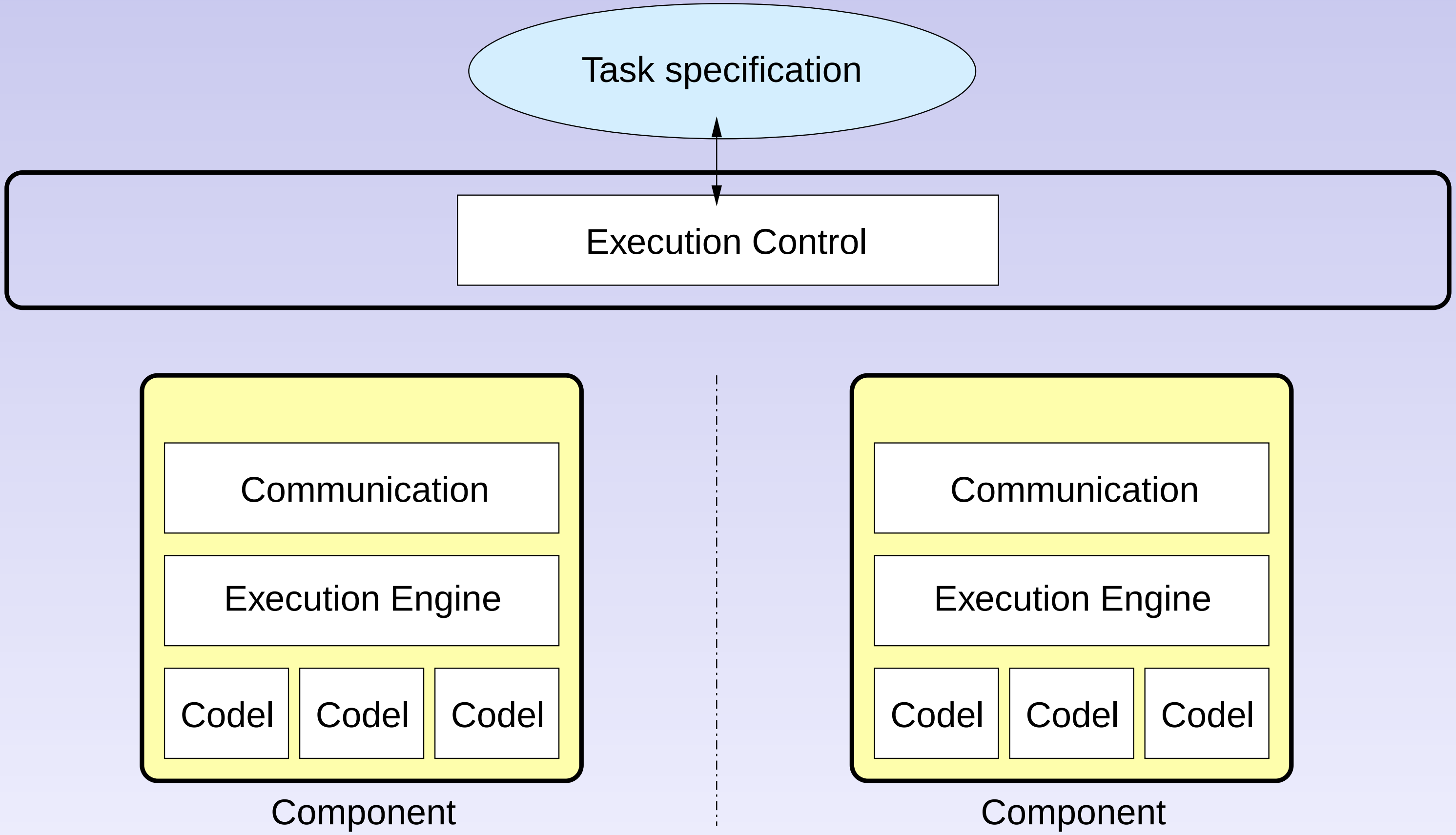
- Deal with robotics constrains, in various domains.
  Real-time, distributed, ...
- Allow the use of various technologies.
  CORBA, XML, programming languages, ...
- Maximize reusability, minimize developments efforts.
- Tackle software validation problems.



Control Flow

Control Flow: Service requests (execution of algorithms or configuration).

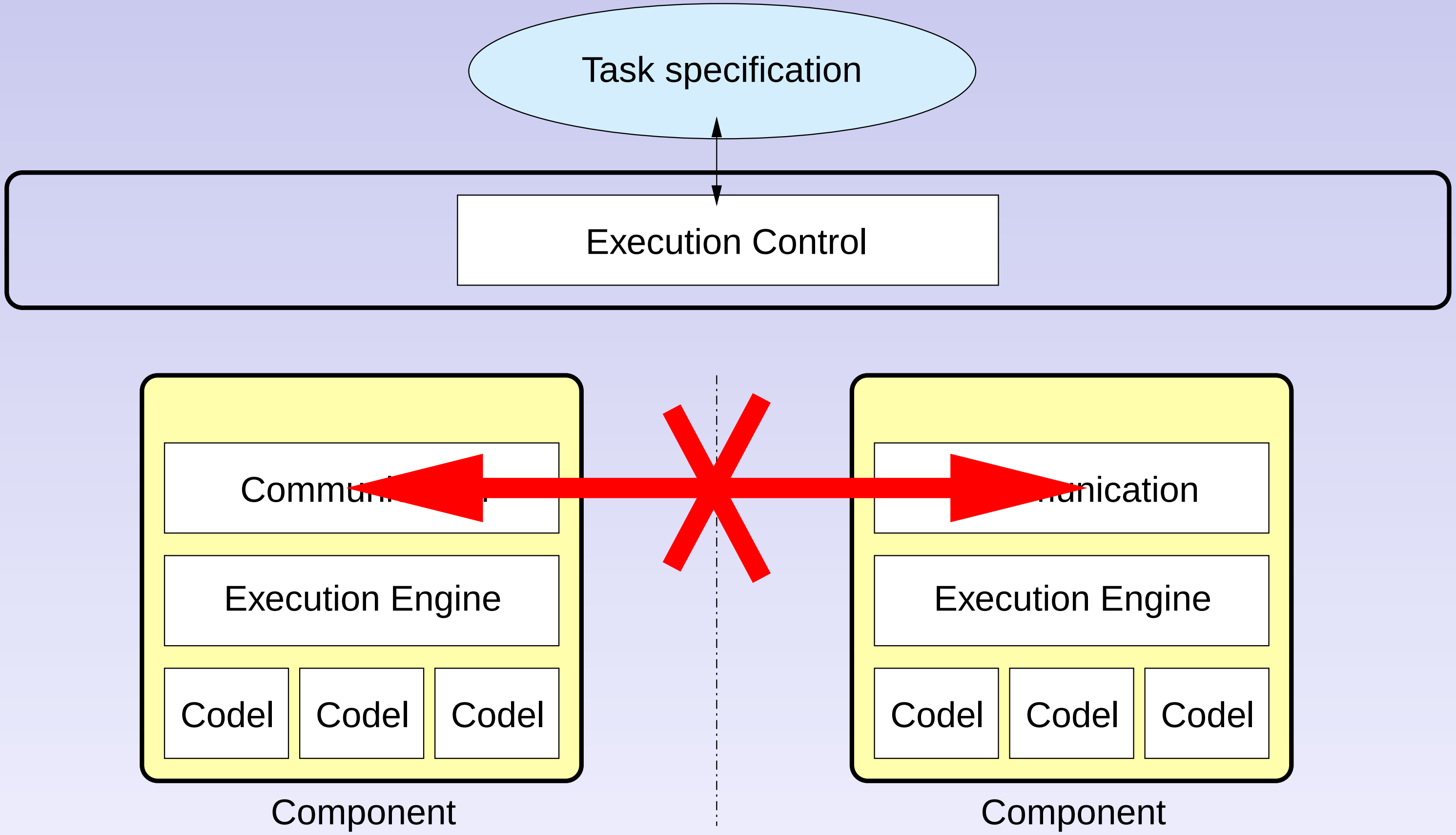
- Established *outside* the components.
- Decoupling between functions and decision.



Control Flow

Control Flow: Service requests (execution of algorithms or configuration).

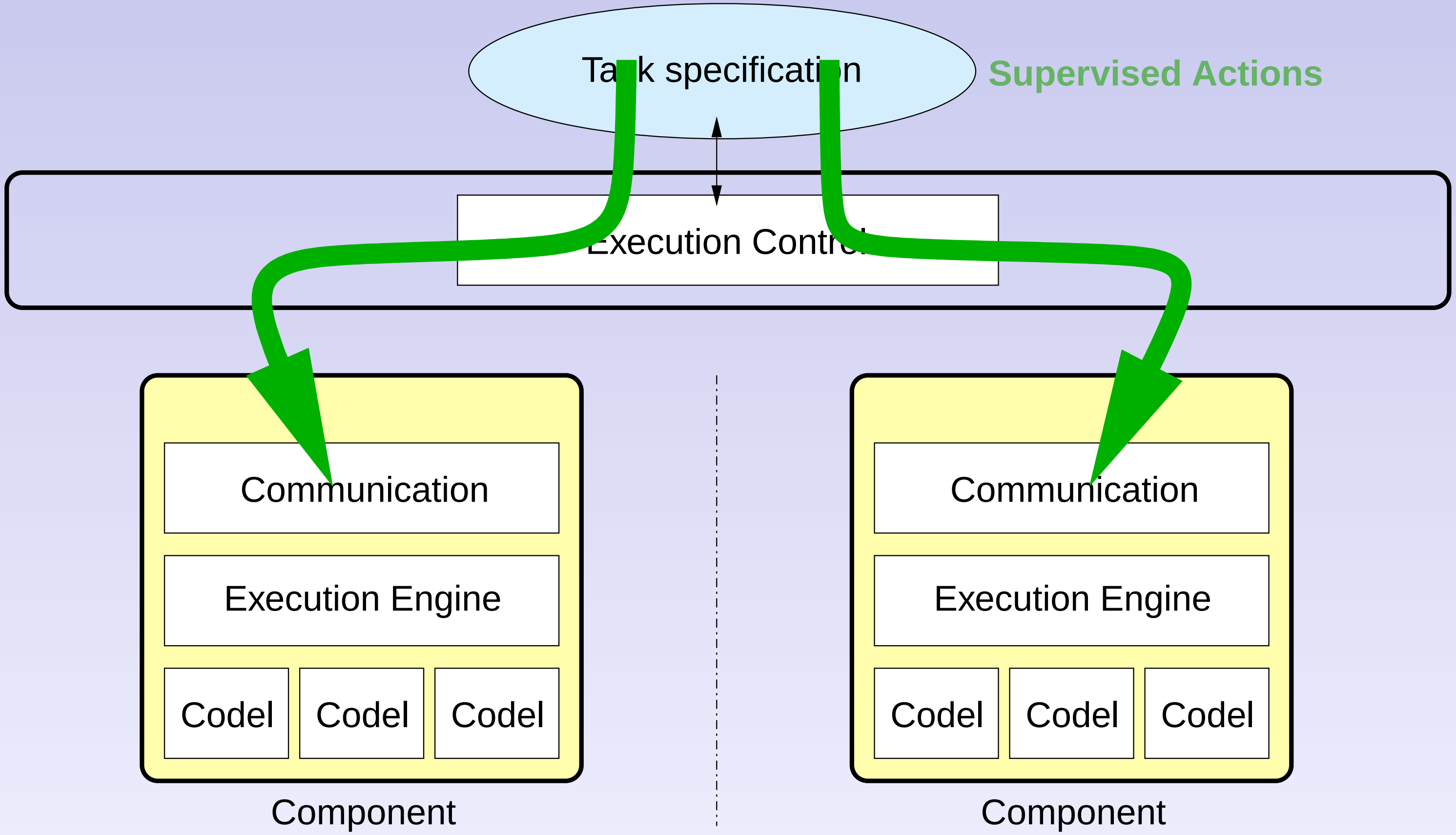
- Established *outside* the components.
- Decoupling between functions and decision.



Control Flow

Control Flow: Service requests (execution of algorithms or configuration).

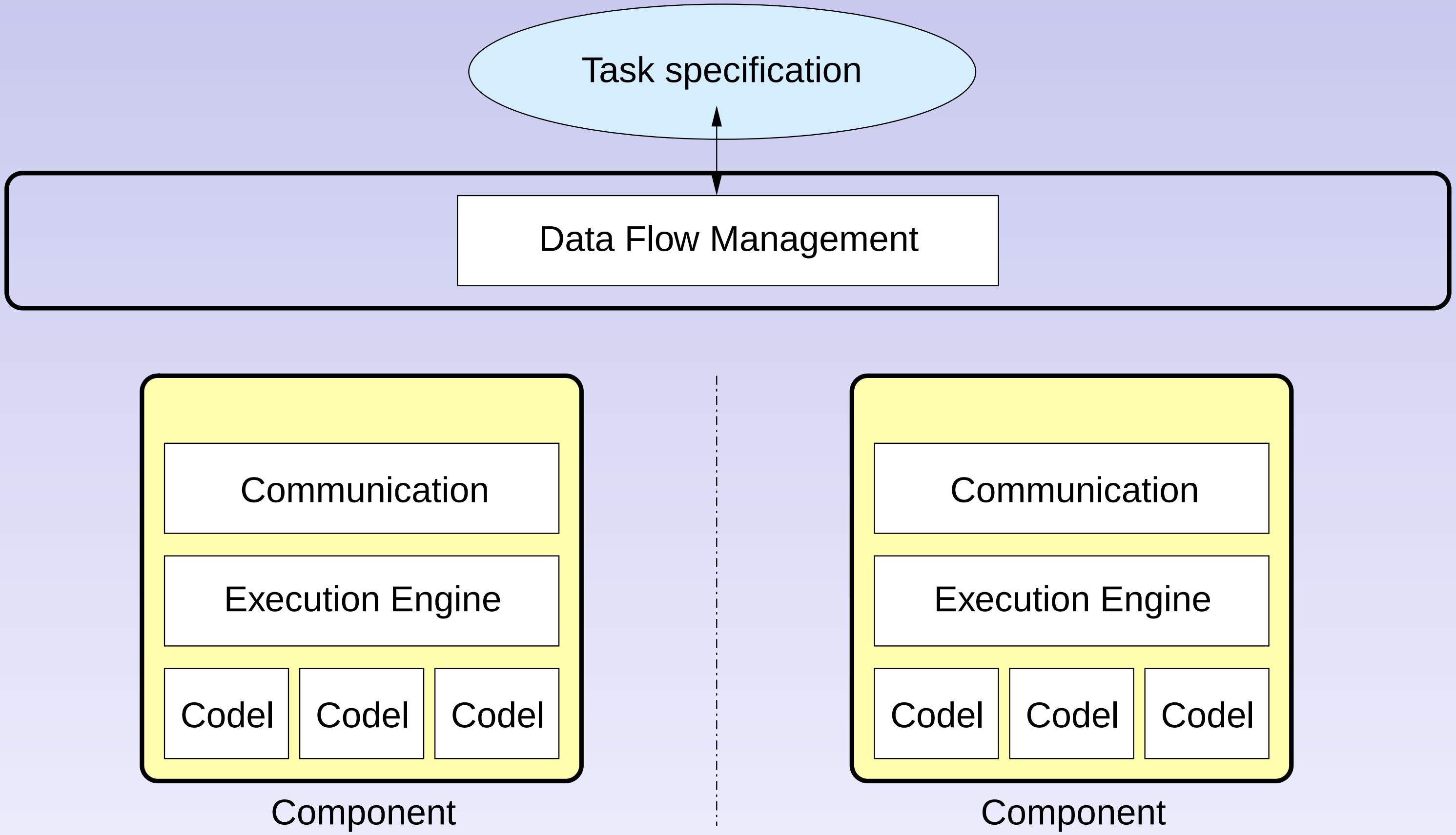
- Established *outside* the components.
- Decoupling between functions and decision.



Data Flow

Data Flow: Connection between components.

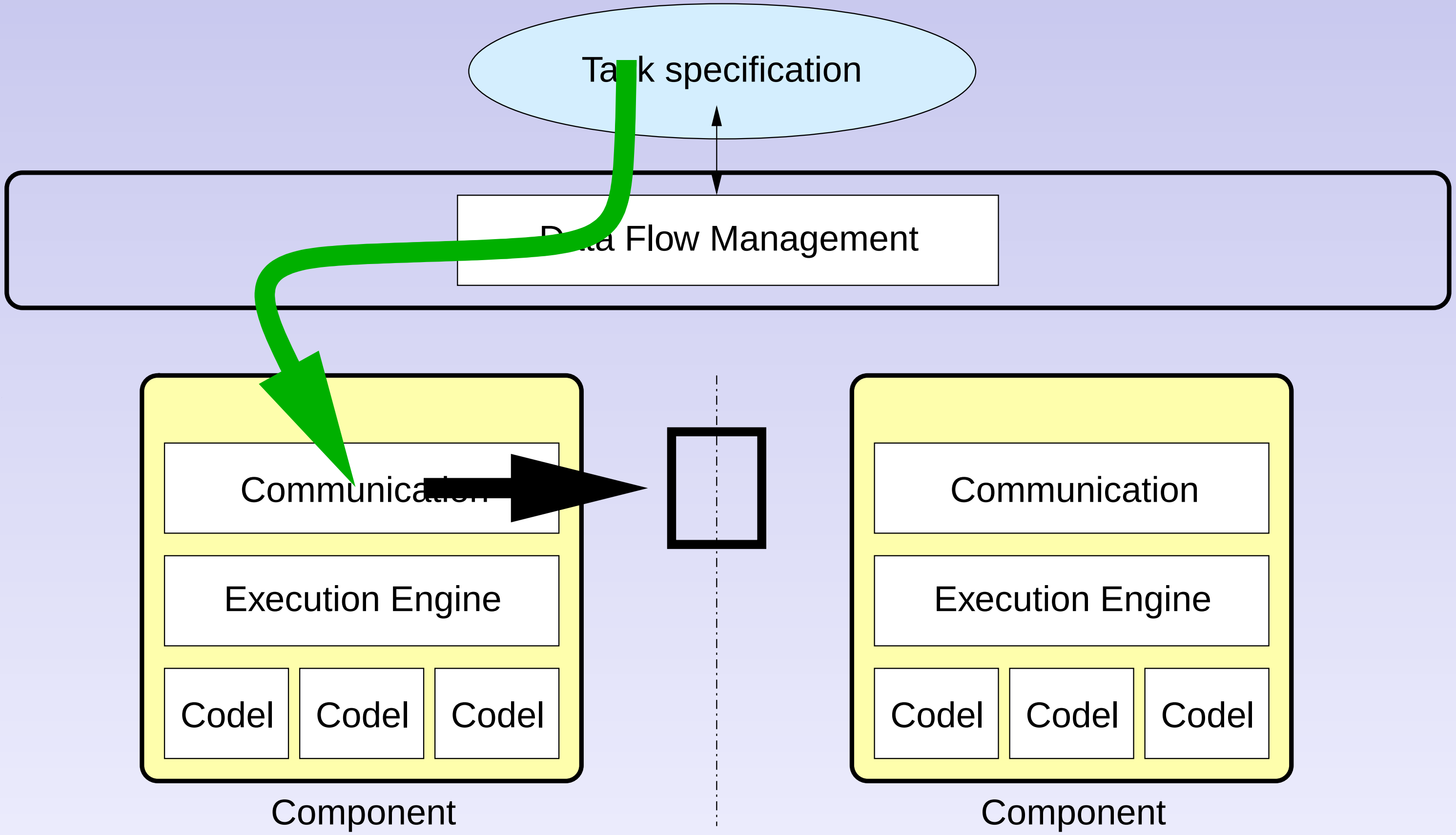
- Programmable by requests, dynamically reconfigurable.



Data Flow

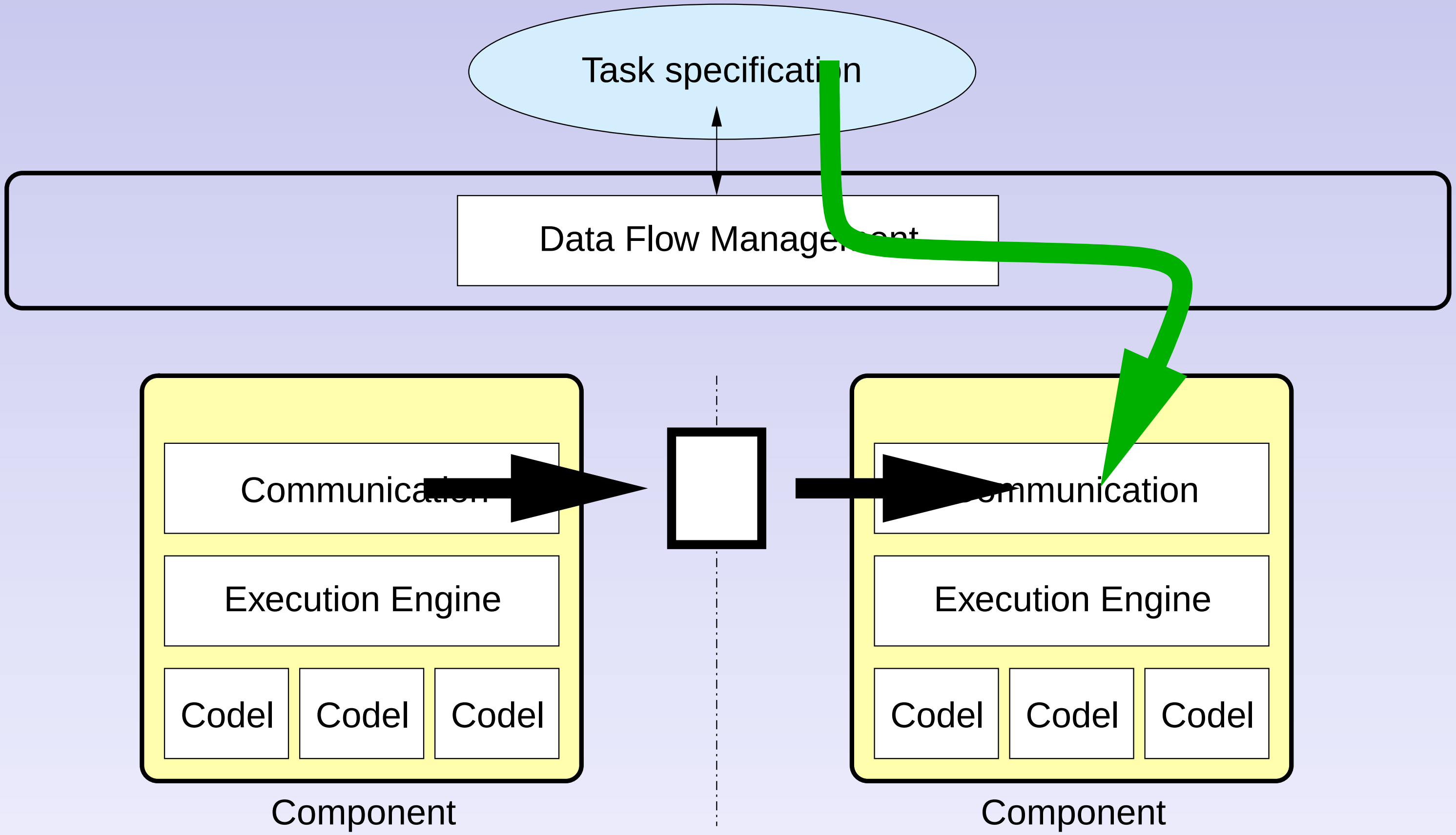
Data Flow: Connection between components.

- Programmable by requests, dynamically reconfigurable.



Data Flow: Connection between components.

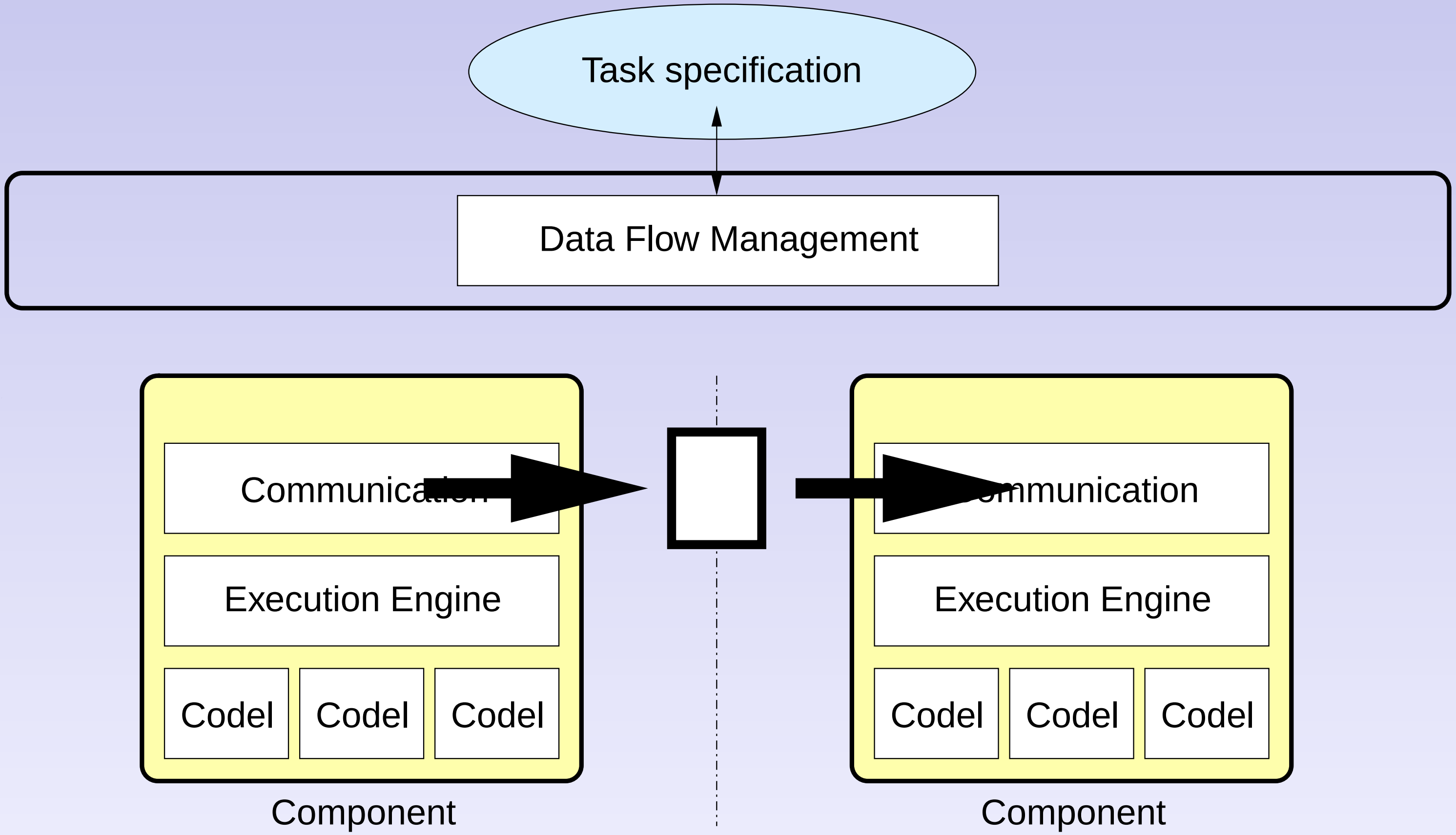
- Programmable by requests, dynamically reconfigurable.



Data Flow

Data Flow: Connection between components.

- Programmable by requests, dynamically reconfigurable.



Component Description Language

Purpose of a description language

- Provide a *formal* description of the underlying libraries.
- Allow the integration within components using *different technologies* with no impact on the underlying libraries.

What does the language describe?

- Interfaces (input and output data, services).
- Decomposition of services into a Finite State Machine (codels).
- Links with libraries.

Services

```
service sampleService(in parameter "A parameter",
                      out result   "Result")
{
  doc: "Short service description";

  start(in parameter) {
    run: sampleStartCodel;
    next: sampleCodel, stop;
  }

  exec sampleCodel(inout parameter, out result) {
    run: sampleCodel;
    next: stop;
  }

  stop(in parameter) {
    run: sampleStopCodel;
  }
}
```

Internal Data

```
#include "strucutres.idl" /* IDL type definitions */

import {
   double               a;
   ComplexStruct        b;
}

parameter {
   sequence<long,10> parameter;
}

private {
   long                 internalData;
}

export {
   double               result[2];
}
```

Threads

```
thread sampleThread {
  clock: internal; /* different sequencing mechanisms */
  clock: external;
  clock: none;

  start() {
    run: threadInit;
  }

  stop() {
    run: threadEnd;
  }
}
```

Task specification

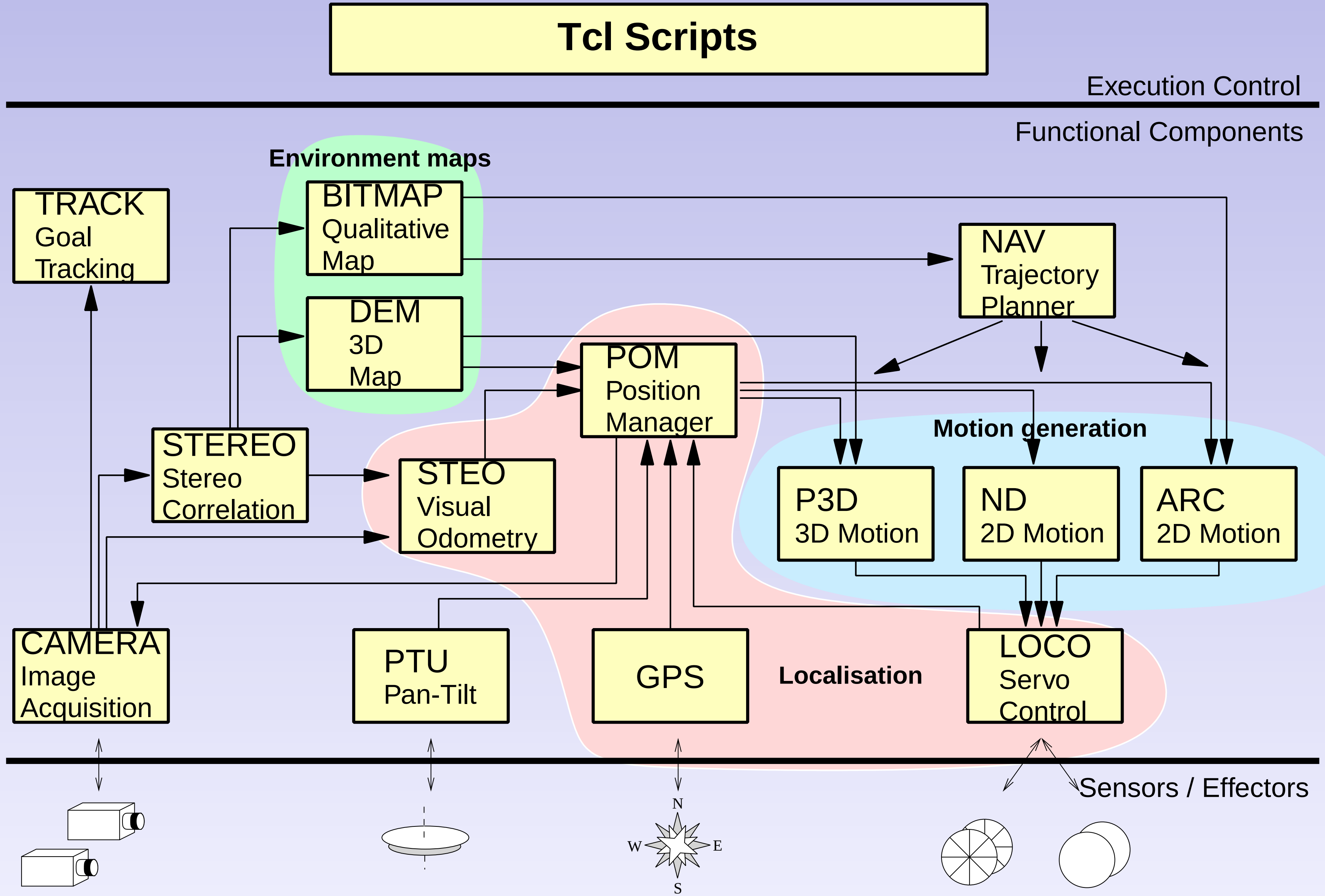
- TCL language interpreter.
- Augmented by procedures corresponding to the set of available services.
- Dynamic loading of components.

Example: Terrain mapping with three components

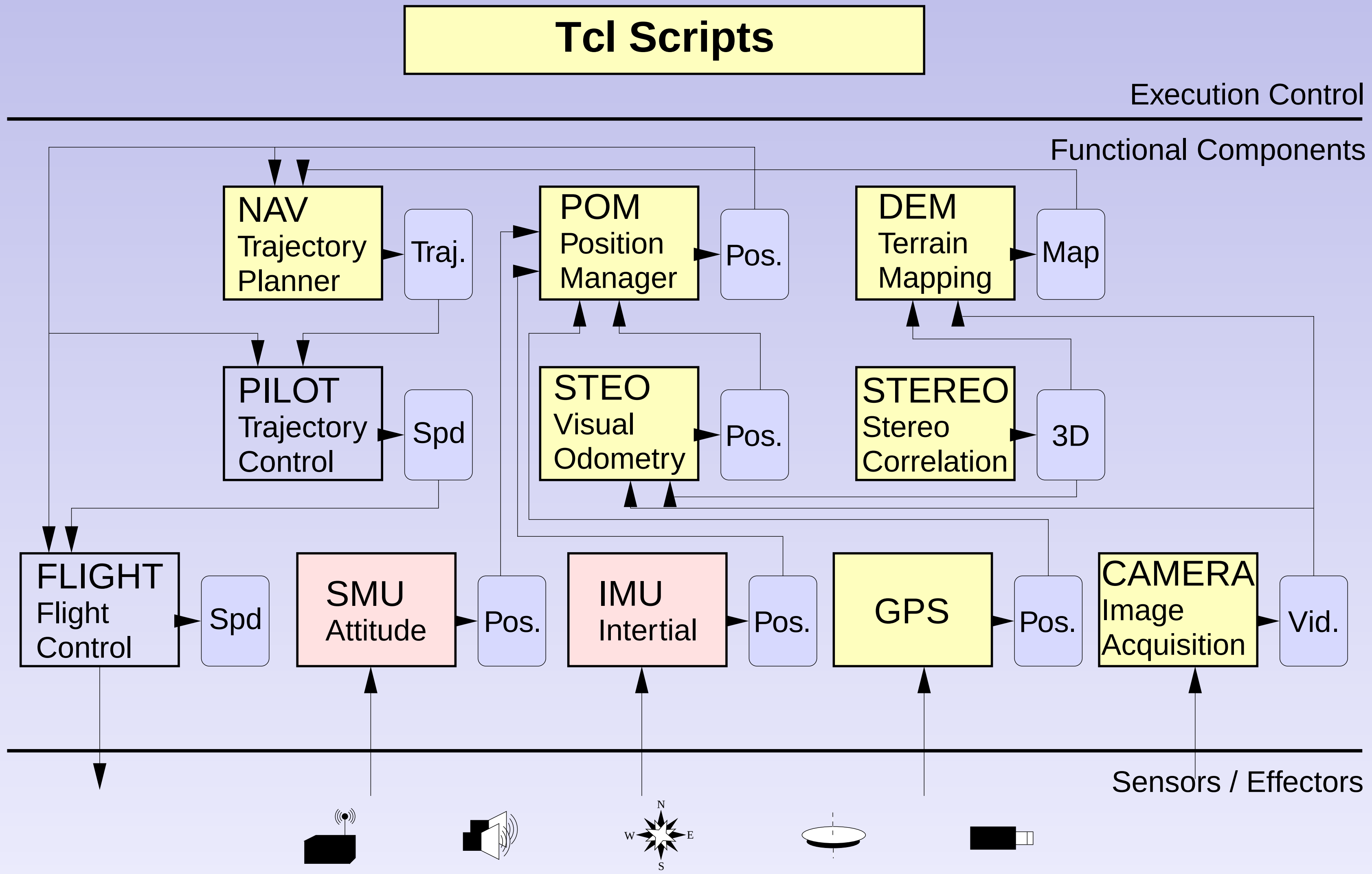
```
# selection of image sources
scorrel::SelectInput CAMERA_BANK_A

# go forward at 5cm/s
loco::GotoSpeed 0.05 0.0

# endless loop with no control!! (illustration purpose only)
while { 1 } {
   # acquire a stereo pair
   camera::OneShot CAMERA_BANK_A CAMERA_STEREO

   # mapping
   scorrel::Compute3DImage
   scorrel::Save3DImage
}
```

Sample Realizations

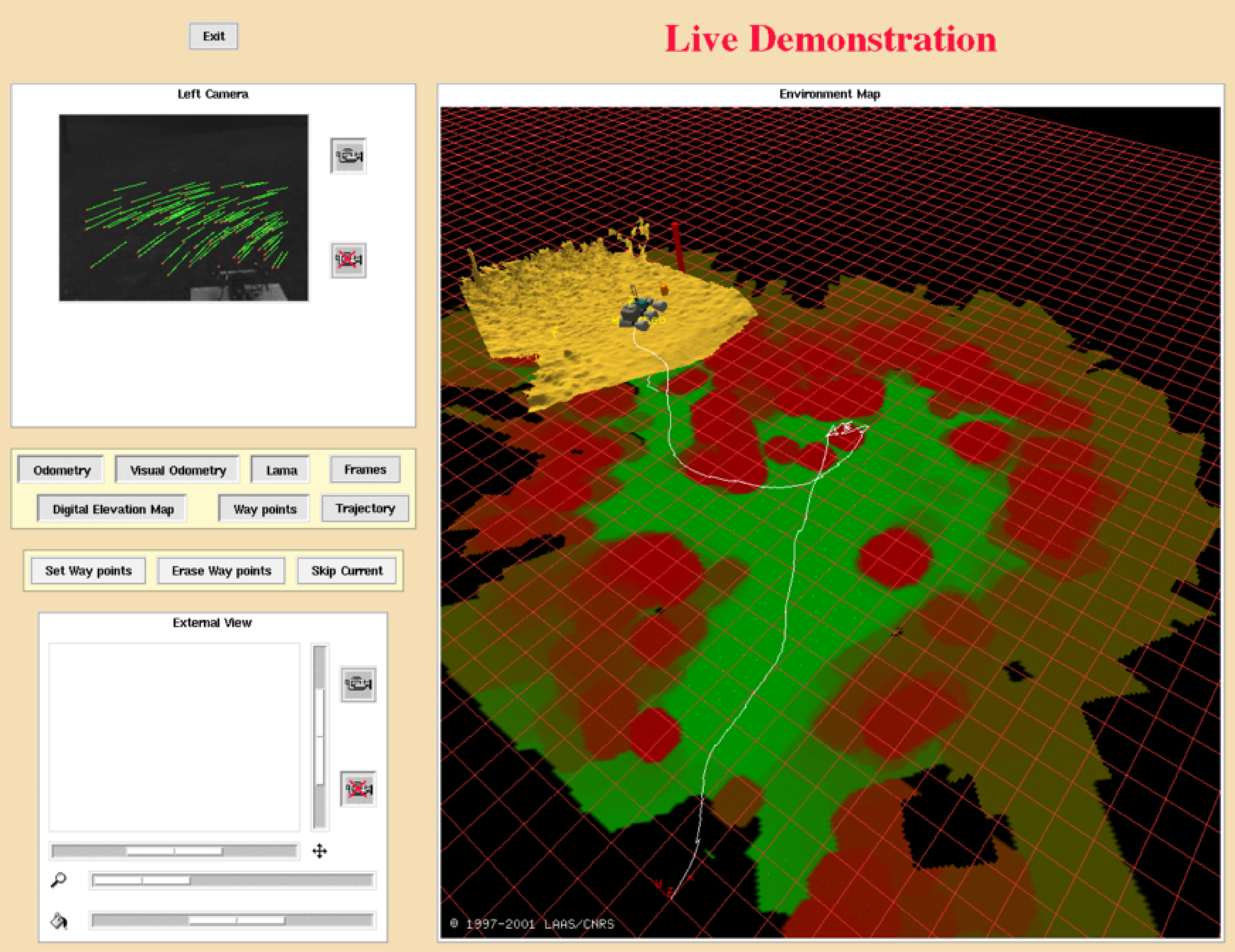
(Lama) Sample Realization



(Karma) Sample Realization



Visualization



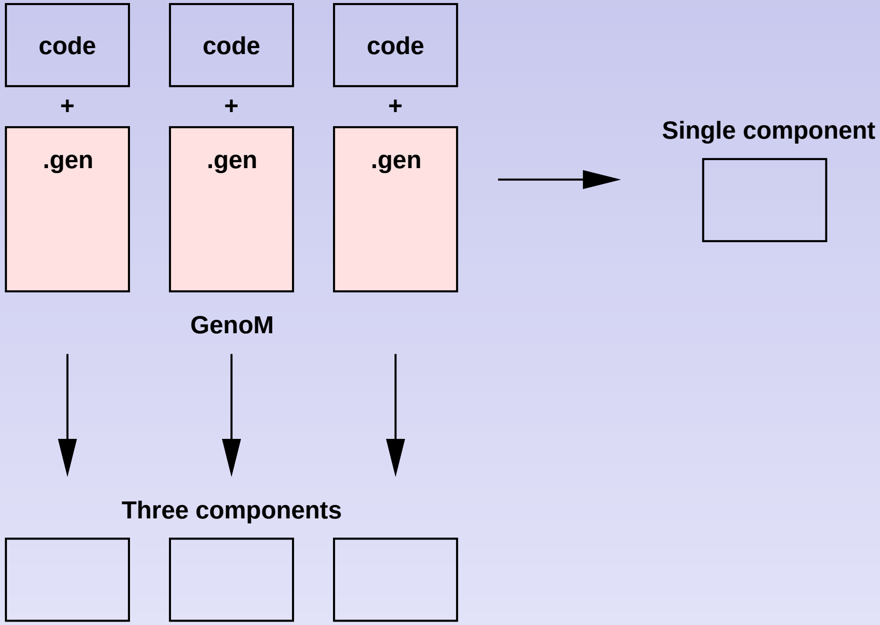
Toward Other Instances

GenoM is a *specific* implementation of the architectural concepts.

Other implementations can be developed.

For instance:

- Different communication strategies.
- Sharing one single execution engine among several components.



Conclusion

- Definition of a component architecture in robotics context.
- Definition of a component description language.
- Complete rework of GenoM to match the new specifications.

Software: http://www.laas.fr/˜mallet/orocos

- Component description parser.
- POSIX Real-Time library (OS abstraction).
- GenoM-1.
- Interactive TCL interpreter for task specification.
- 3D visualization tool.