A Framework for Advanced Robot Programming in the RoboCup Domain

Hayato Kobayashi[1], Akira Ishino[2], and Ayumi Shinohara[3]

[1]Department of Informatics, Kyushu University, Japan
[2]Office for Information of University Evaluation, Kyushu University, Japan
[3]Graduate School of Information Science, Tohoku University, Japan

Outline

- Background
- Related work
- Proposed framework
  - Concept
  - Plug-in system
  - Scripting language
- Demonstrations
- Discussion
- Conclusions and future work

Outline

- Background
- Related work
- Proposed framework
  - Concept
  - Plug-in system
  - Scripting language
- Demonstrations
- Discussion
- Conclusions and future work

Kyushu University and Tohoku University

Team ***Jolly Pochie*** [dʒóli·pótʃi:]

RoboCup Soccer

Small size robot league

Four-legged robot league

Middle size robot league

Simulation league

https://www.robocup.org/

Humanoid league

Many difficulties

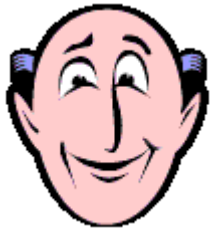
- Soccer programs is complex
- Full compiling takes more than 10 minutes
- Booting of AIBO takes about 30 seconds
- Debug via wireless LAN
- Batteries only last about 30 minutes
- Team development can cause conflictions
- Cute shape is not suited for playing soccer
- AIBOs can faint because of motor load
- AIBOs can break their legs

How to solve?

- Hardware problems

We can’t solve!

- Software problems



Maybe, we can solve!!!

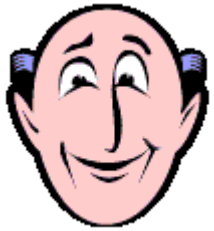
How to solve?

- Hardware problems

We can’t solve!

- Sof

We want a “framework”
that makes it easy to create robot programs



Maybe, we can solve!!!

Outline

- Background
- Related work
- Proposed framework
  - Concept
  - Plug-in system
  - Scripting language
- Demonstrations
- Discussion
- Conclusions and future work

Related Work

- Tekkotsu
  - A development framework for AIBO
  - Created at Carnegie Mellon University
  - Consists of C++ Libraries wrapping OPEN-R.

(We must use OPEN-R for creating AIBO programs)

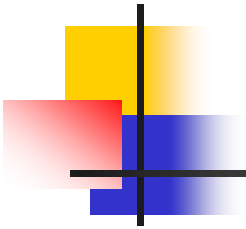
http://www.tekkotsu.org/

Related Work in RoboCup Symposium

- Alessandro Farinelli, Giorgio Grisetti, and Luca Iocchi. “**SPQR-RDK: A Modular Framework for Programming Mobile Robots**”. In *RoboCup 2004: Robot Soccer World Cup VIII*, LNAI, pages 660--653. Springer, 2005.
- Alexander Kleiner and Thorsten Buchheim. “**A Plugin-Based Architecture for Simulation in the F2000 League**”. In *RoboCup 2003: Robot Soccer World Cup VII*, LNAI, pages 434--445. Springer, 2004.
- Thomas Röfer. “**An Architecture for a National RoboCup Team**”. In *RoboCup 2002: Robot Soccer World Cup VI*, LNAI, pages 417--425. Springer, 2003.
- Paul A. Buhler and Jose M. Vidal. “**Biter: a Platform for the Teaching and Research of Multiagent Systems' Design using RoboCup**”. In *RoboCup 2001: Robot Soccer World Cup V*, LNAI, pages 299--304. Springer, 2002.

Outline

- Background
- Related work
- Proposed framework
  - Concept
  - Plug-in system
  - Scripting language
- Demonstrations
- Discussion
- Conclusions and future work



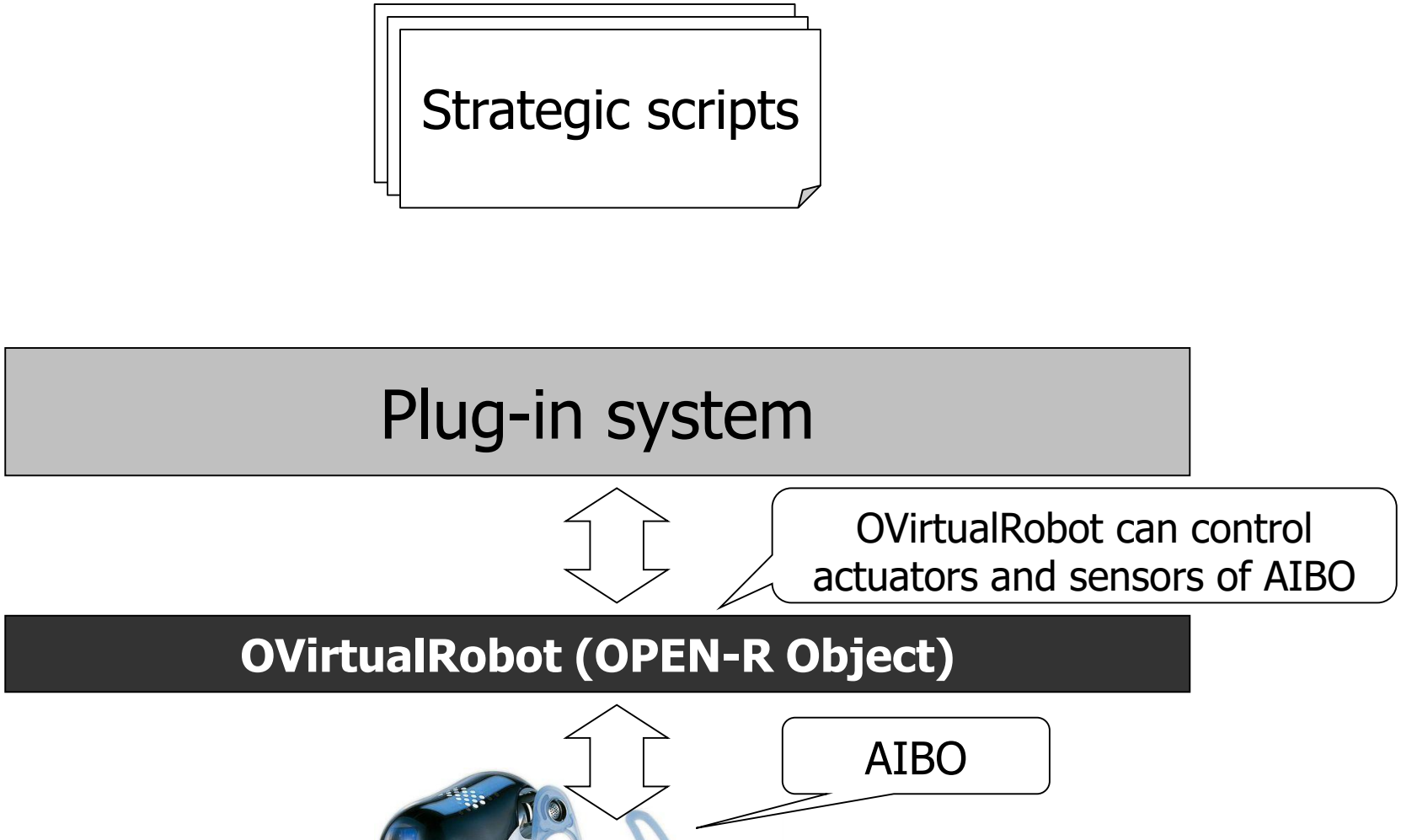
Proposed Framework

- Two techniques are integrated
  - **Plug-in system** (easy to extend)
    - Effective for team development
    - No need to know the whole system
  - **Scripting language** (easy to use)
    - Effective for creating strategic programs
    - No need to recompile and reboot

Outline

- Background
- Related work
- Proposed framework
  - Concept
  - Plug-in system
  - Scripting language
- Demonstrations
- Discussion
- Conclusions and future work

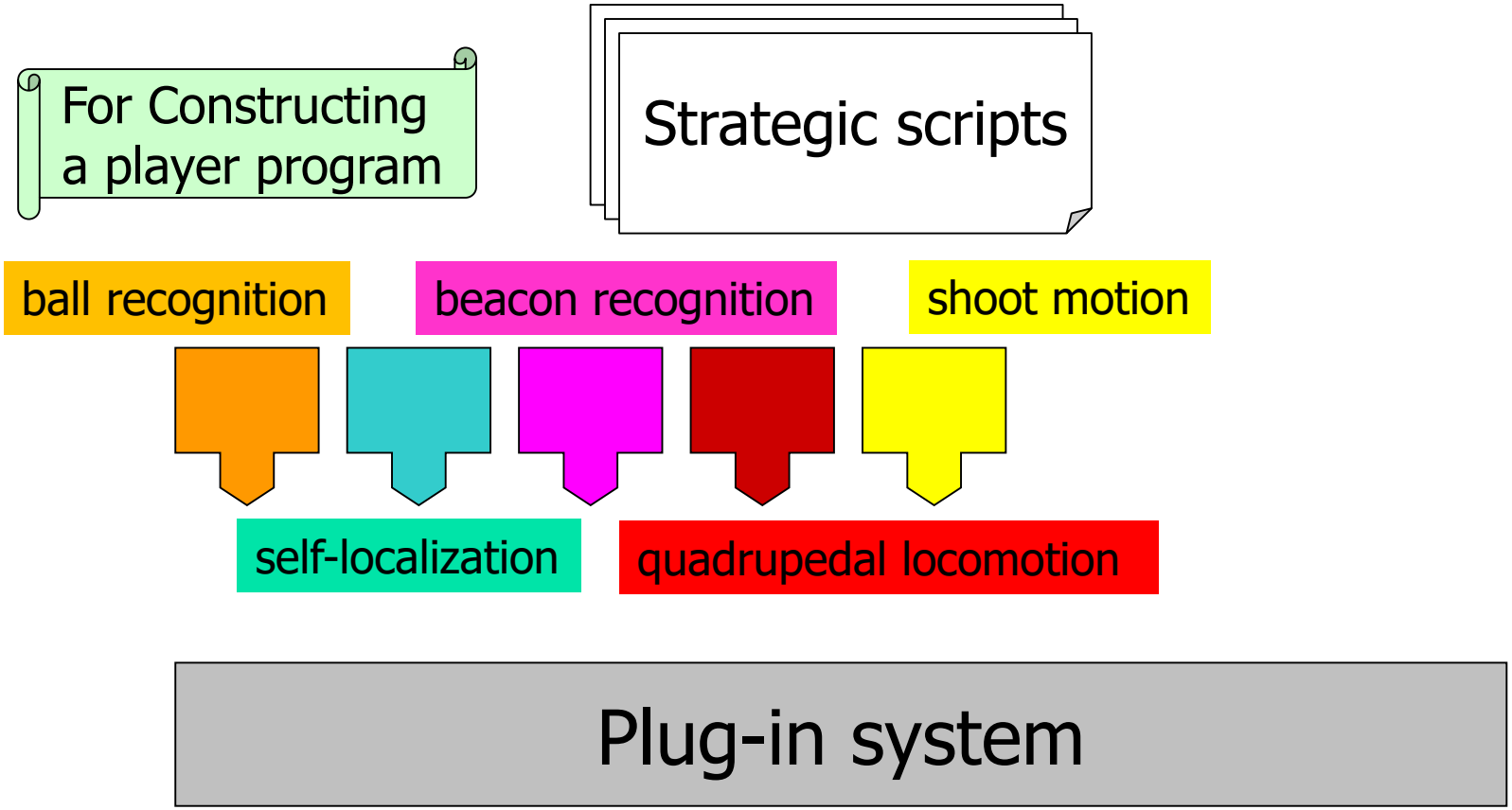
Concept of our framework



Concept of our framework

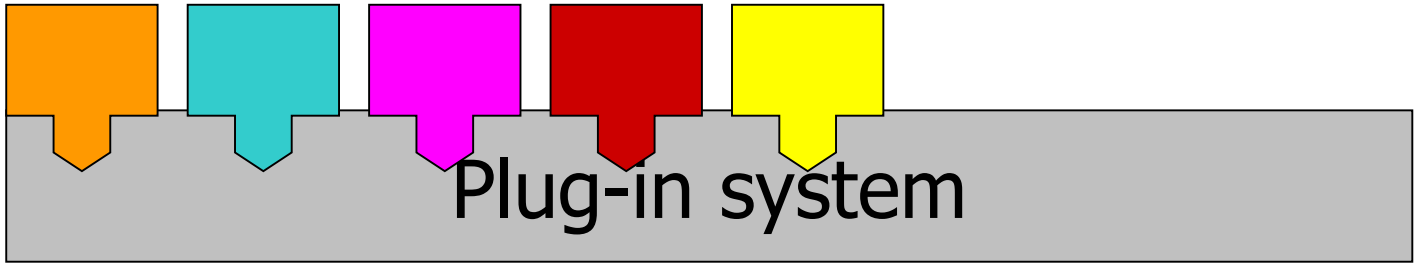
Plug-in system

Concept of our framework

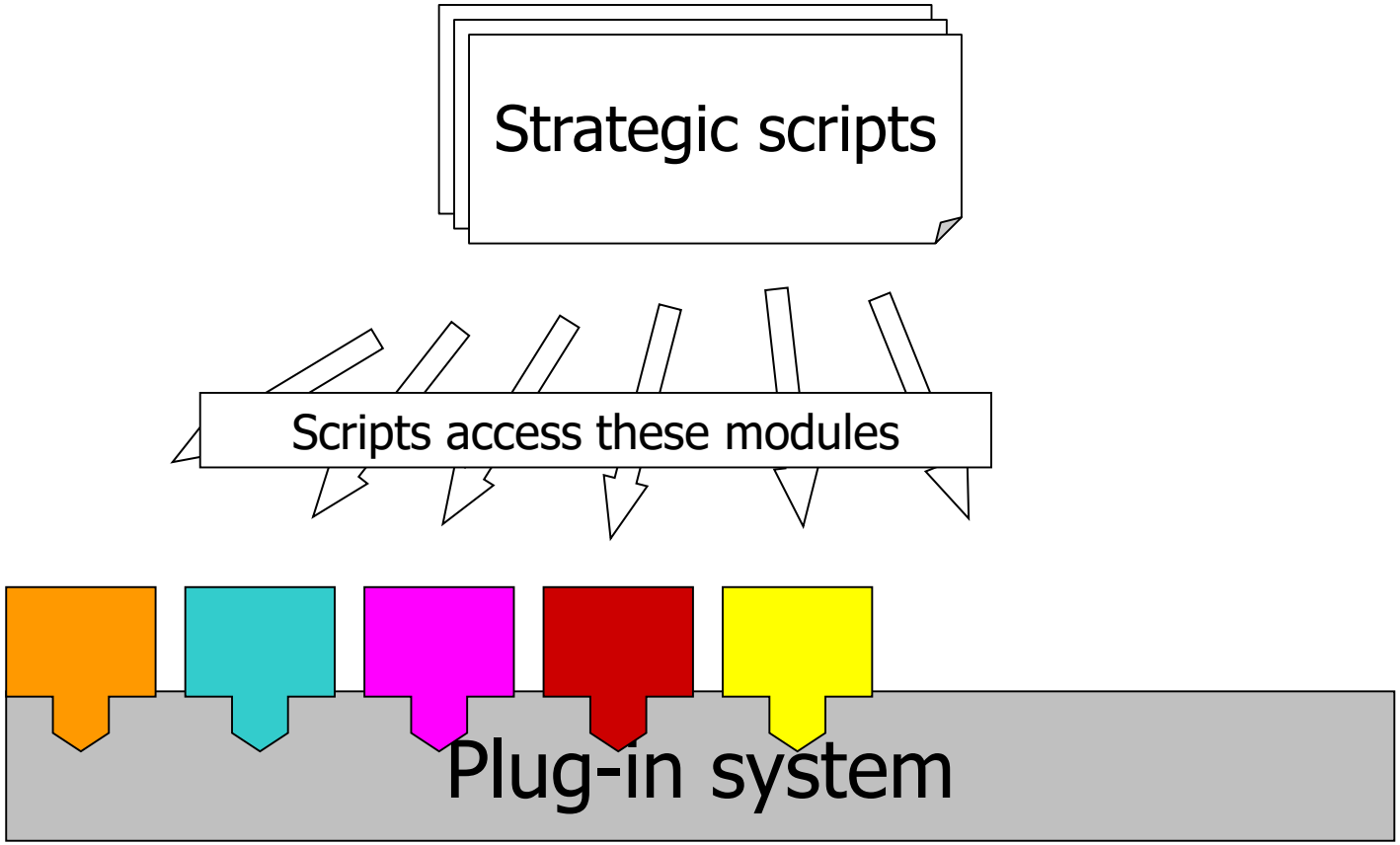


Concept of our framework

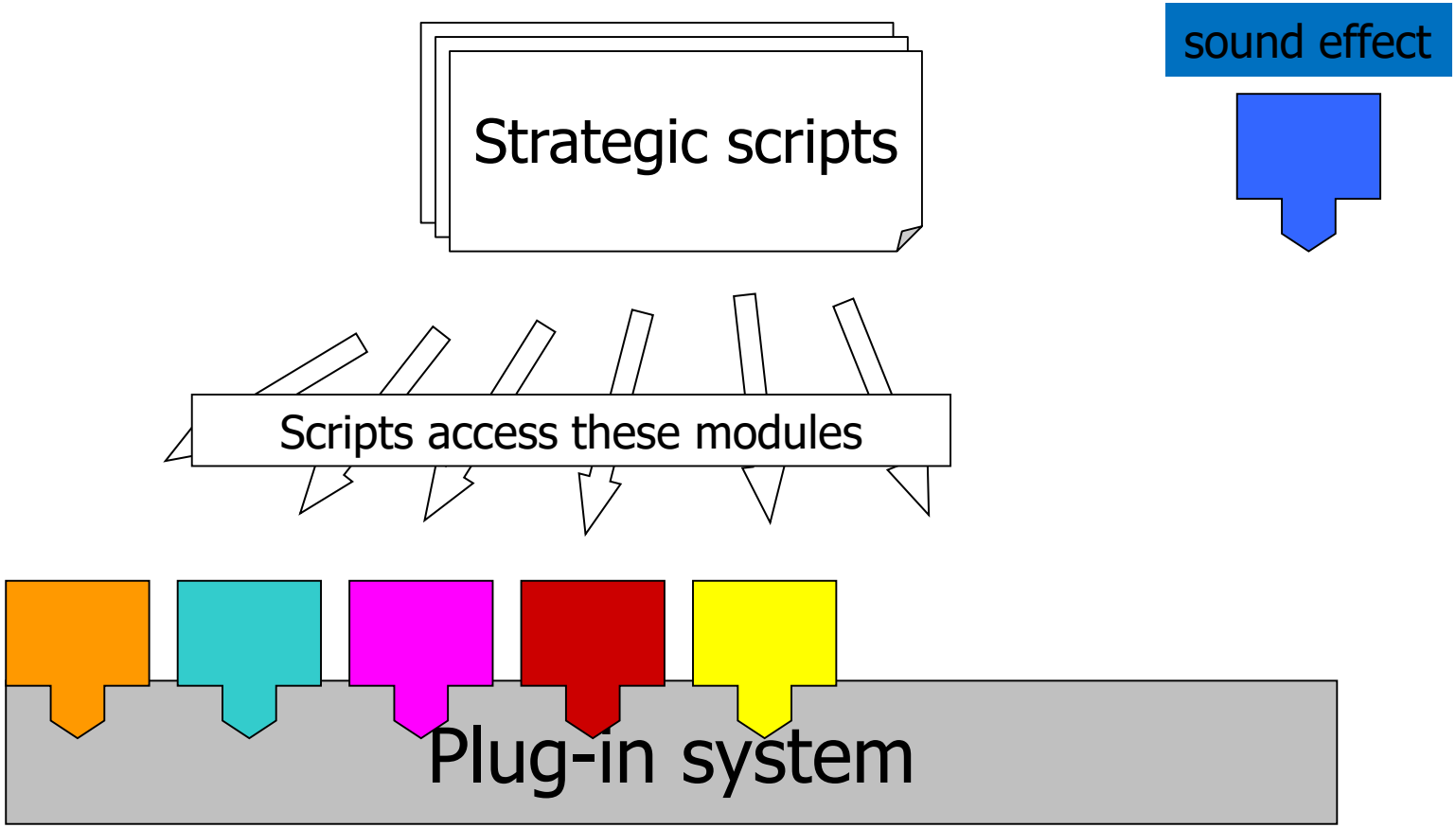
Easy to plug



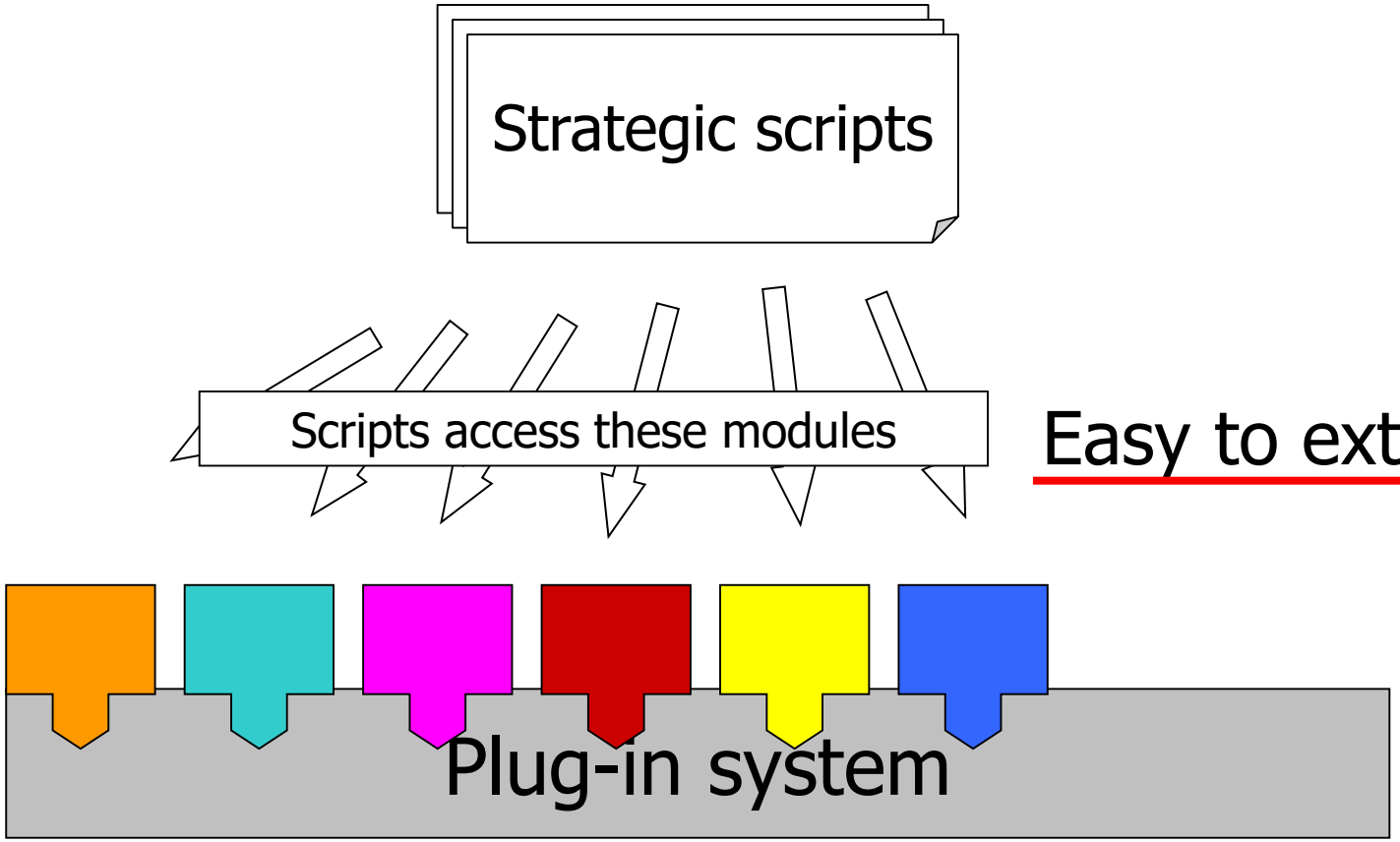
Concept of our framework



Concept of our framework

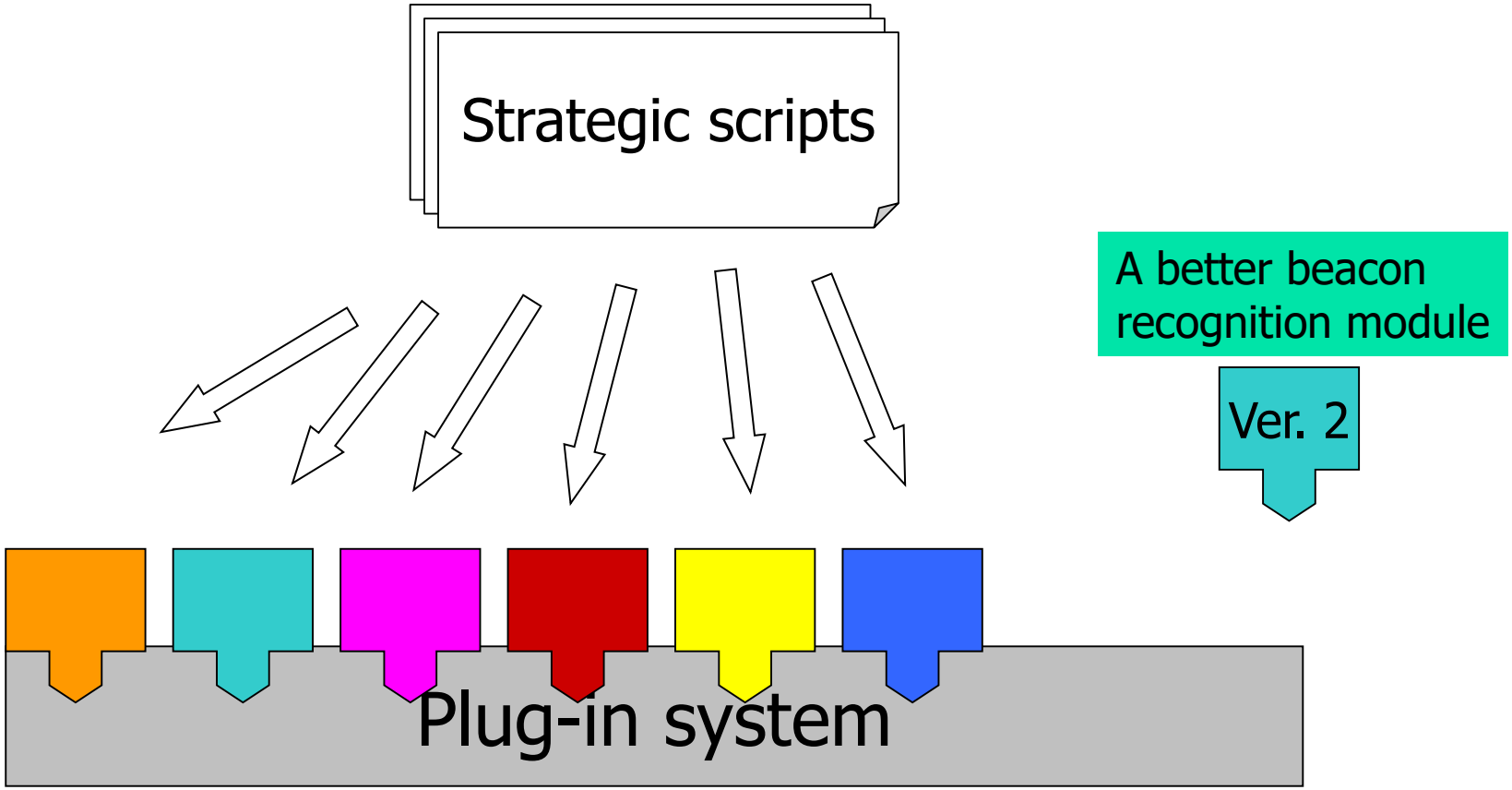


Concept of our framework

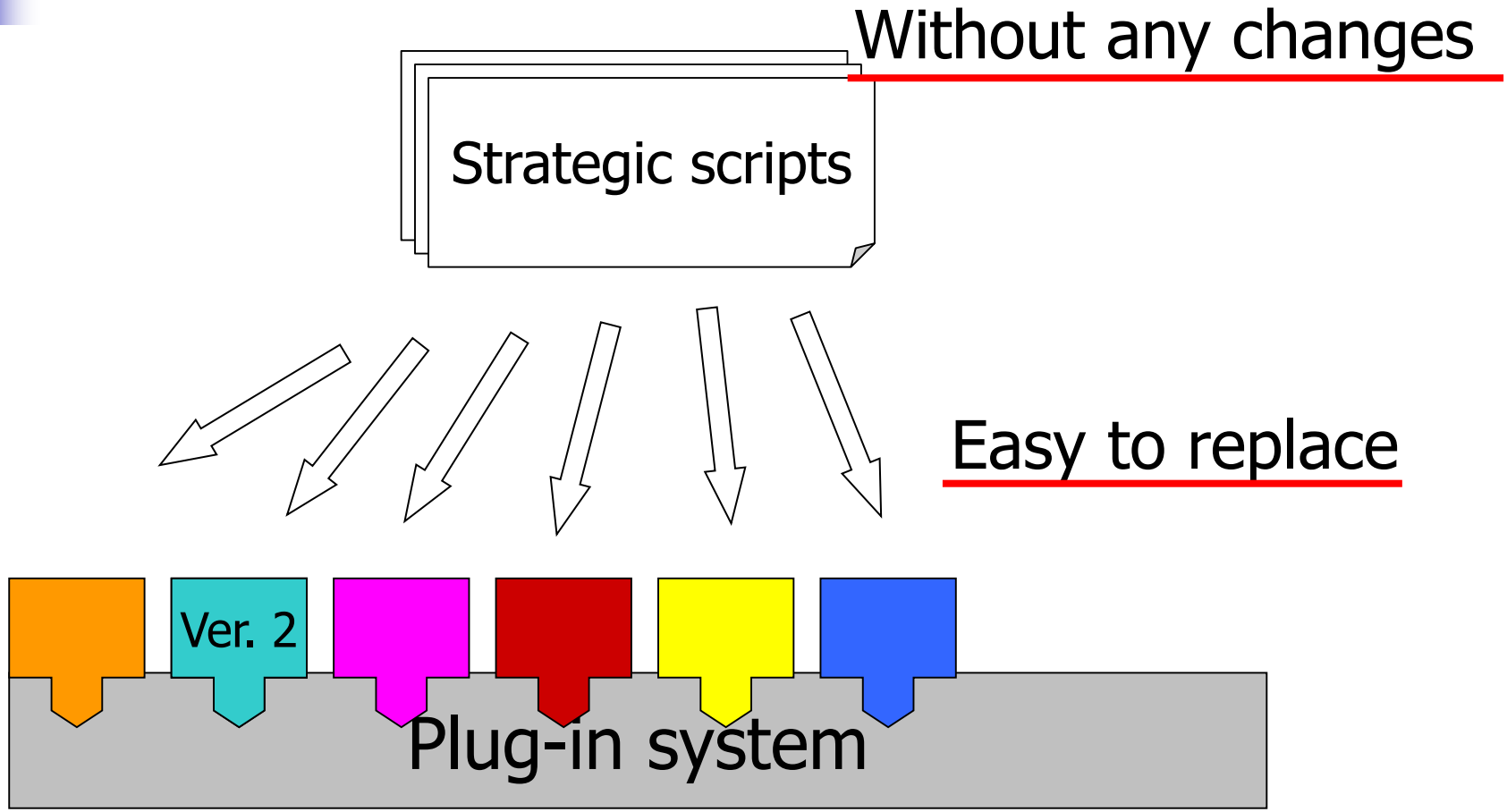


Easy to extend

Concept of our framework



Concept of our framework



Outline

- Background
- Related work
- Proposed framework
  - Concept
  - Plug-in system
  - Scripting language
- Demonstrations
- Discussion
- Conclusions and future work

Plug-in system

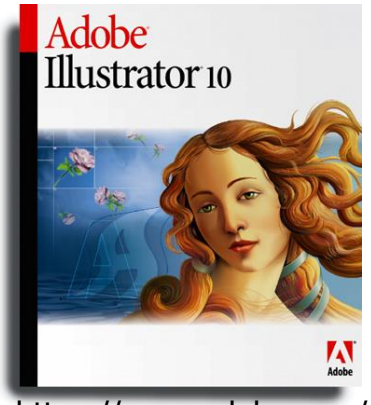
- Plug-in system has often been used in recent applications

https://www.mozilla.org/

Web browser

https://www.eclipse.org/

IDE

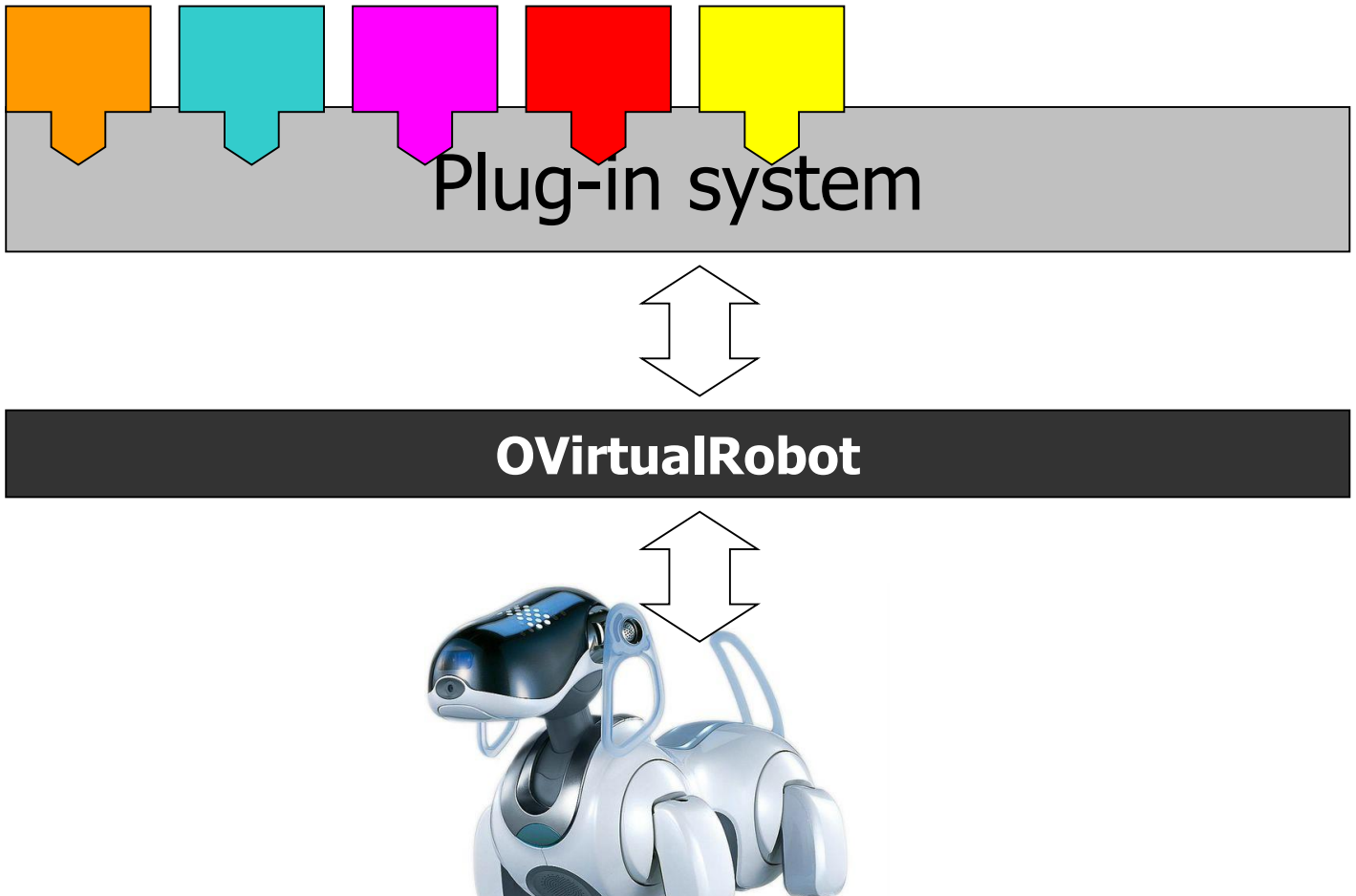


https://www.adobe.com/

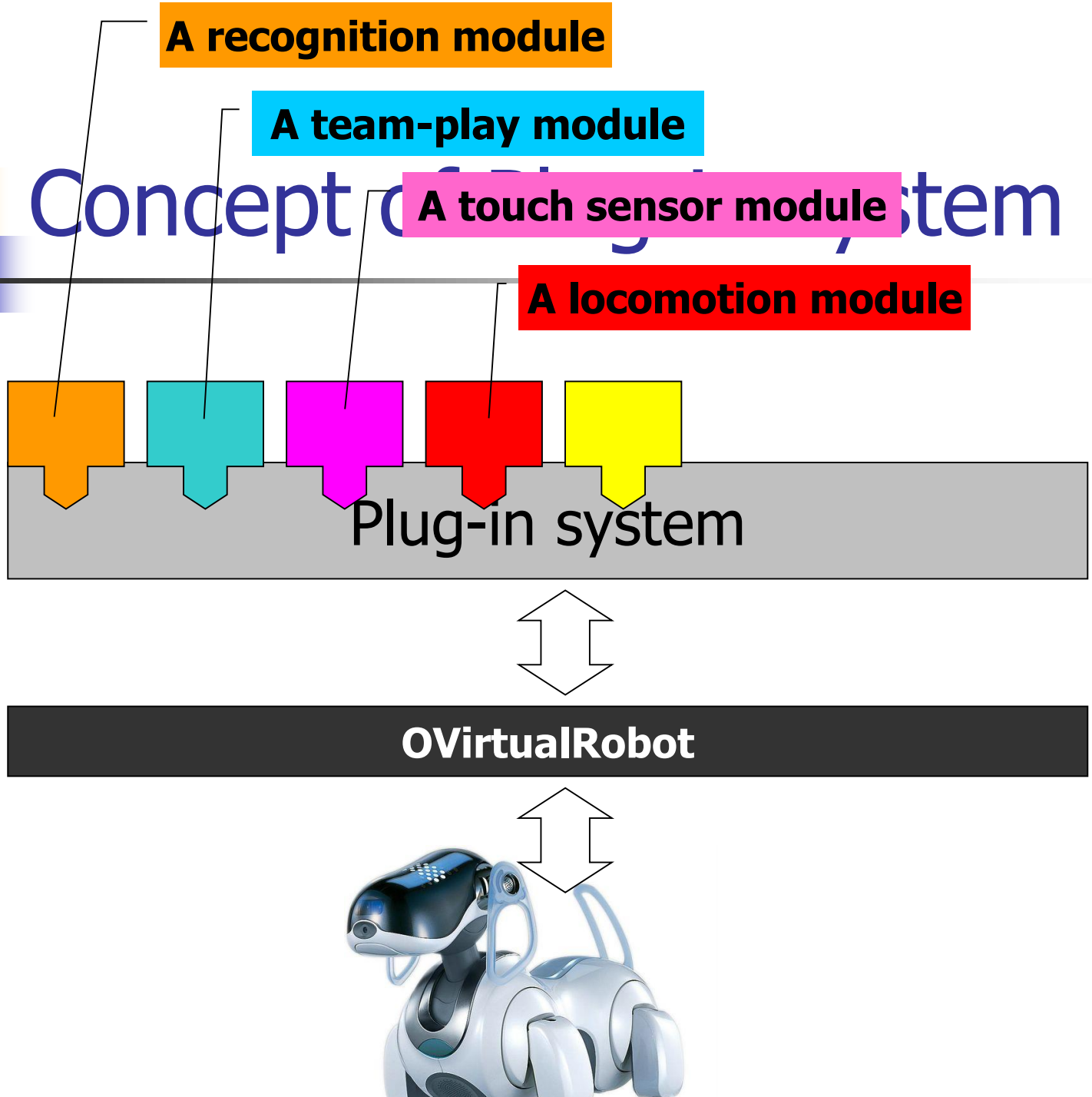
Drawing software

We don’t need to know the whole system

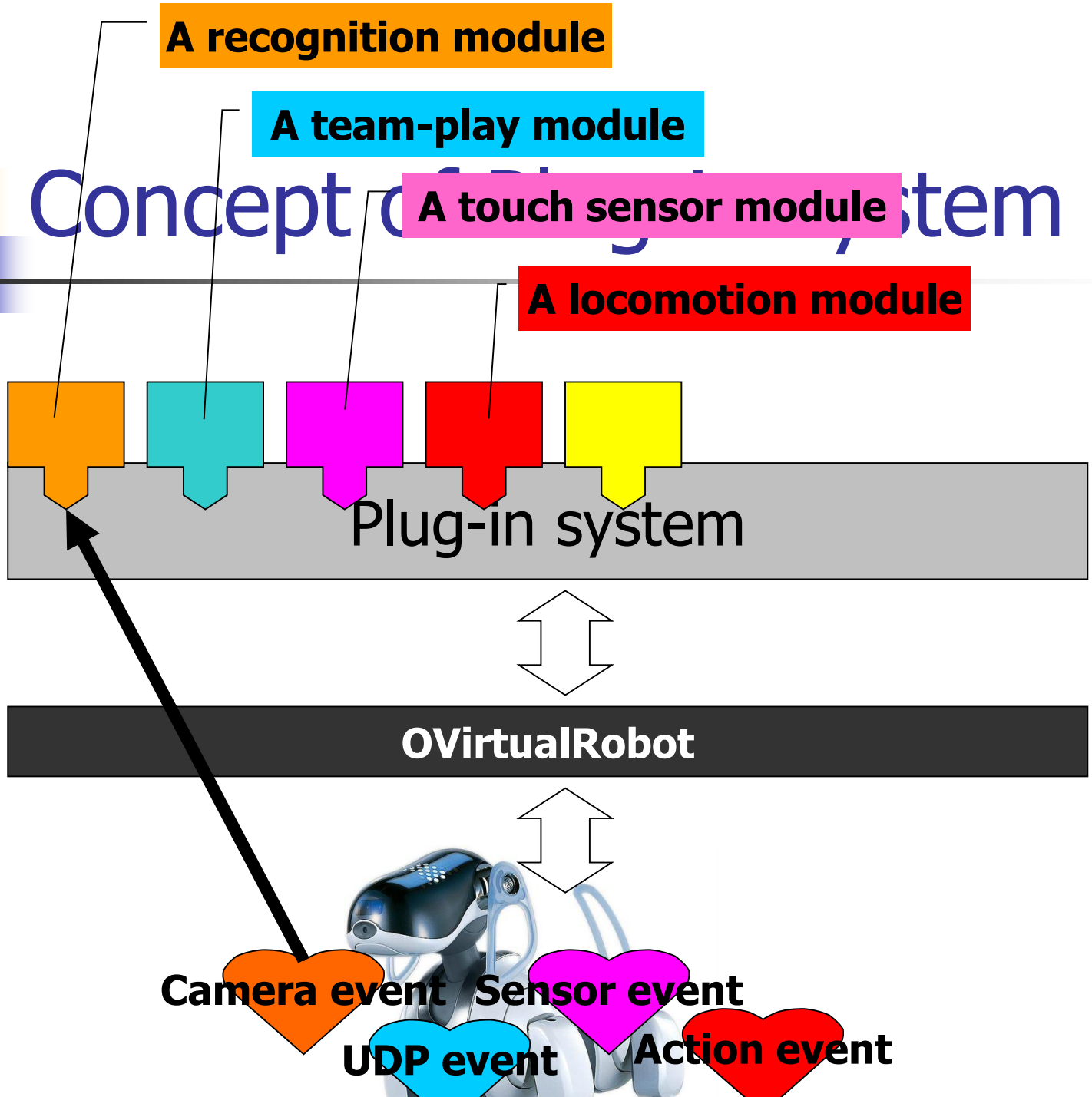
Concept of Plug-in system



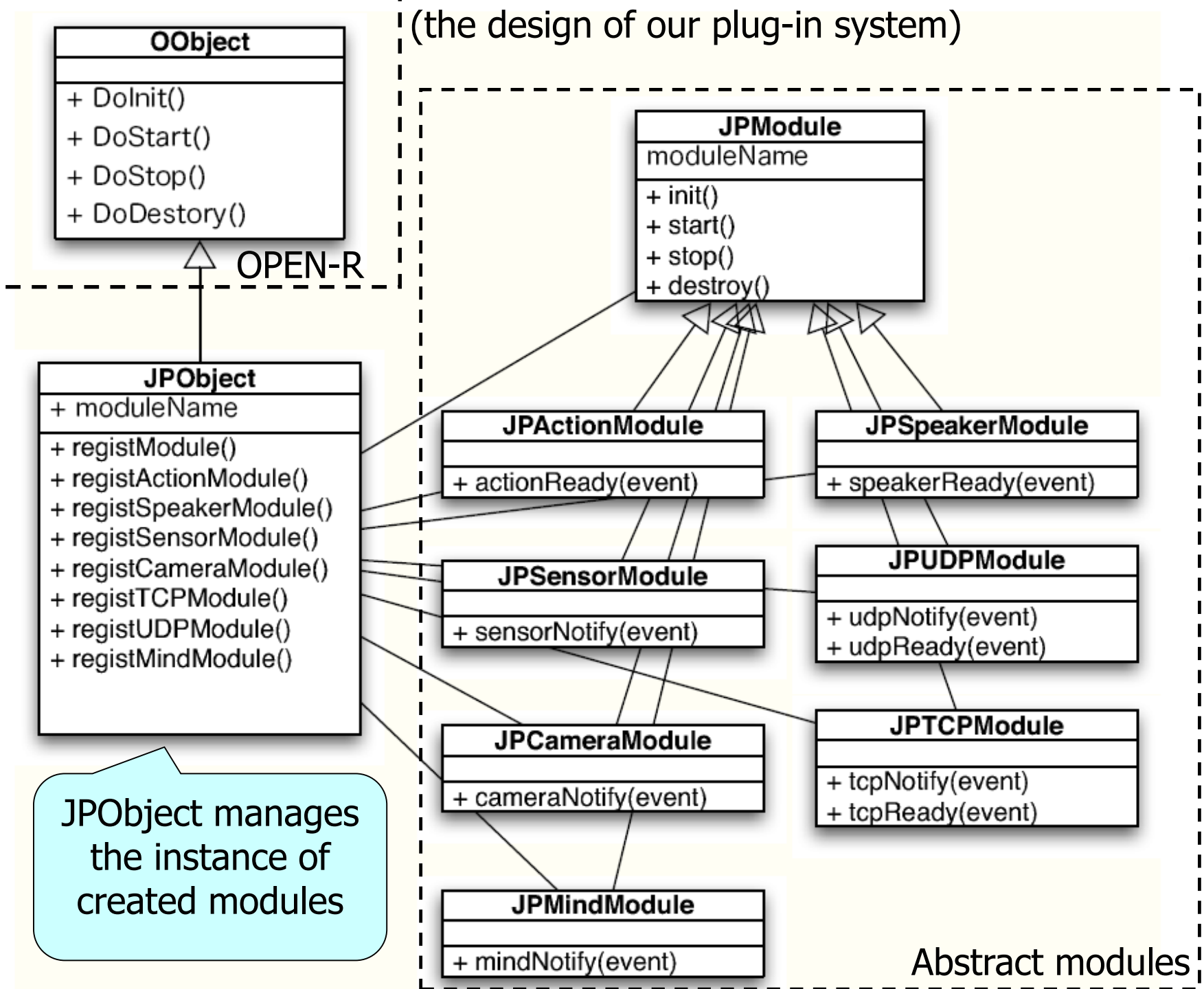
Concept of Plug-in system



Concept of Plug-in system



(the design of our plug-in system)



Abstract modules

We can override cameraNotify() and describe image processing

Abstract Module	Special method	When is the method called?
JPCameraModule	cameraNotify()	Every 40 ms in sync with the CCD-camera
JPMindModule	mindNotify()	The same as cameraNotify()
JPActionModule	actionReady()	When a set of joint angles are achieved
JPSensorModule	sensorNotify()	When sensor data is detected
JPUDPModule	udpNotify()	When UDP data is received
JPTCPModule	tcpNotify()	When TCP data is received
JPModule		

Abstract modules

recognition modules

strategy modules

locomotion modules

localization modules, etc

Abstract Module	Sp[illegible] method	When is the method called?
JPCameraModule	[illegible]	[illegible] the [illegible]
JPMindModule	[illegible]NotifyO()	The same as cameraNotify()
JPActionModule	[illegible]	[illegible] e achieved
JPSensorModule	sensorNotify()	When sensor data is detected
JPUDPModu[illegible]	[illegible]	[illegible] received
JPTCPModule	[illegible]y()	[illegible] is received
JPModule		

Outline

- Background
- Related work
- Proposed framework
  - Concept
  - Plug-in system
  - Scripting language
- Demonstrations
- Discussion
- Conclusions and future work

Lua (http://www.lua.org/)

- Designed for embedding into C/C++
  - Easy to embed into C/C++
  - Faster than Python
  - Uses less memory than Python
  - Has a simple and powerful syntax
  - Smaller footprint than Python (about 1/10)

By http://lua-users.org/wiki/LuaVersusPython

A simple example

- Returns the summation of arguments

```
function sum(…)
   local s = 0
   for i=1, arg.n do
         s = s + arg[i]
   end
   return s
end
```

Luabind (http://www.luabind.sourceforge.net/)

- A library that helps us create bindings between C++ and Lua
  - Utilizing template meta programming, we can easily register C++ functions and call Lua functions

An example of Binding in modules

- For lua scripts to use C/C++ functions

```
void
BasicMotion6JPM::init() {
    module(JPLua::L) [
        class_<BasicMotion6JPM>("BasicMotion6JPM")
        .def("swingHead", &BasicMotion6JPM::swingHead)
        .def("stopSwingHead", &BasicMotion6JPM::stopSwingHead)
        …..
    ];
    get_globals(JPLua::L)["basicMotion"] = this;
}
```

An example of Binding in modules

- For lua scripts to use C/C++ functions

```
void
BasicMotion6JPMinit() {
                                                    M")
                                                    gHead)
    .def("stopSwingHead", &BasicMotion6JPM::stopSwingHead)
    …..
  ];
  get_globals(JPLua::L)["basicMotion"] = this;
}
```

basicMotion:swingHead(0,0,0)

(in Lua scripts)

An example of robot script

- Swings its head from side to side

```
require "JPLib/Syslog.lua"
require "JPLib/Units.lua"
require "JPLib/STree.lua"

function init()
  stree:setState("swingLeft")
end

function mindNotify()
  stree:doAction()
end
```

```
function swingLeft()
  basicMotion:swingHead(0,d2ur(80),0,”swingRight”)
  stree:setState(“swingWait”)
end
function swingRight()
  basicMotion:swingHead(0,d2ur(-80),0,”swingLeft”)
  stree:setState(“swingWait”)
end
```

Other teams

- **MicroPerl** by team *UPennalizers*
- **Python** by team *rUNSWift* and *CMDash*
- **Scheme** by team *ASURA*

need

Definition of global wrapping functions for binding to C/C++

Binding of Python

```
/* Get the project for ball. When this function is called,
   it is assumed that the robot can see the ball. */
static PyObject * VisionLink_getProjectedBall(PyObject * /*self*/, PyObject *args)
    // Track visual ball.
    int inpoints[2] = {((int) (vision->vob[vobBall].cx) - WIDTH / 2), -1 * ((int) (visi
>vob[vobBall].misc) - HEIGHT / 2)};
    double outpoints[2] = {-1, -1};
    vision->projectPoints(inpoints, 1, outpoints, 0);
    double ballx = -outpoints[0];
    double bally = outpoints[1];

    PyObject *t;
    t = PyTuple_New(2);
    PyTuple_SetItem(t, 0, PyFloat_FromDouble(ballx));
    PyTuple_SetItem(t, 1, PyFloat_FromDouble(bally));
    return t;
}
```

Our choice

- **Lua** & **Luabind** library

need

Simple function calls for binding to C/C++

An example of Binding in modules

- For lua scripts to use C/C++ functions

```
void
BasicMotion6JPM::init() {
    module(JPLua::L) [
        class_<BasicMotion6JPM>("BasicMotion6JPM")
        .def("swingHead", &BasicMotion6JPM::swingHead)
        .def("stpwingHead", &BasicMotion6JPM::stopSwingHead)
        …..
    ];
     get_globals(JPLua::L)["basicMotion"] = this;
}
```

Outline

- Background
- Related work
- Proposed framework
  - Concept
  - Plug-in system
  - Scripting language
- <u>Demonstrations</u>
- Discussion
- Conclusions and future work

Our programming method

Create modules by C++

Need to compile

Create a binary

Need to send the binary
and reboot AIBO

Create a script by Lua

Need to only send the script

Test on AIBO

Our programming method

Create modules by C++

Need to compile

Create a binary

Need to send the binary and reboot AIBO

Create a script by Lua

Need to only send

Test on AIBO

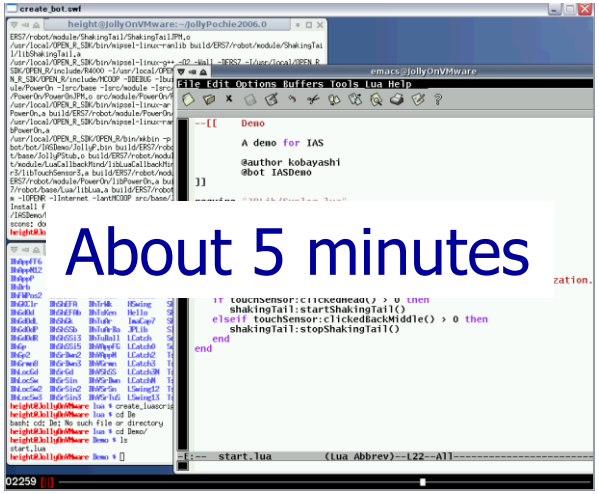
For professionals

For beginners

Demonstrations

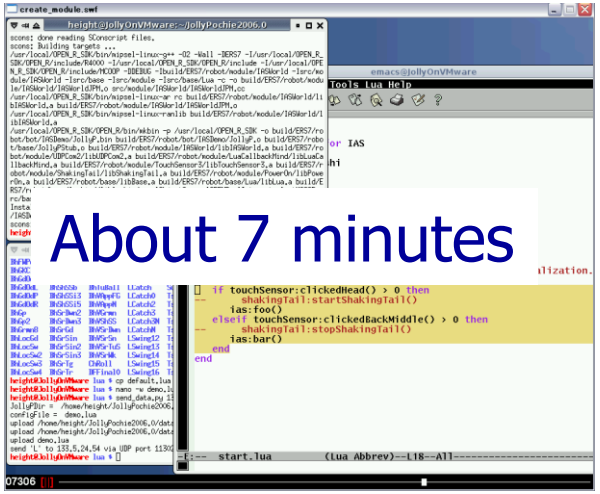
- Available online at following URL

https://youtu.be/mehBFIsW4lQ



Create a binary and script

https://youtu.be/_EgW_0Isx8U



Create a module and bind it

Outline

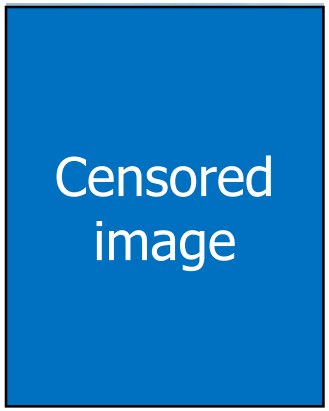
- Background
- Related work
- Proposed framework
  - Concept
  - Plug-in system
  - Scripting language
- Demonstrations
- Discussion
- Conclusions and future work

Discussion

- Rough Comparison of working efficiency
  - In RoboCup 2004 (before using our framework)
  - In RoboCup 2005 (after using our framework)

Experimental result?

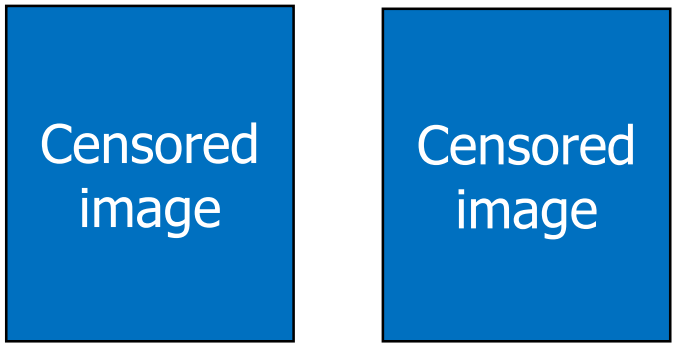
- Me as a beginner (in RoboCup 2004)
  - 3 days to run a sample program
  - 6 months to create a player program for actual games



me
(I was an undergraduate student)

Experimental result?

- Beginner A and B (in RoboCup 2005)
  - 1 hour to run a sample program
  - 1 month to create a player program for actual games



Beginner A Beginner B

(they are still undergraduate students)

Jolly Pochie in RoboCup 2005

- 130 modules and 350 scripts
- Top 8 in Soccer competition
- 7th in Technical Challenge

Outline

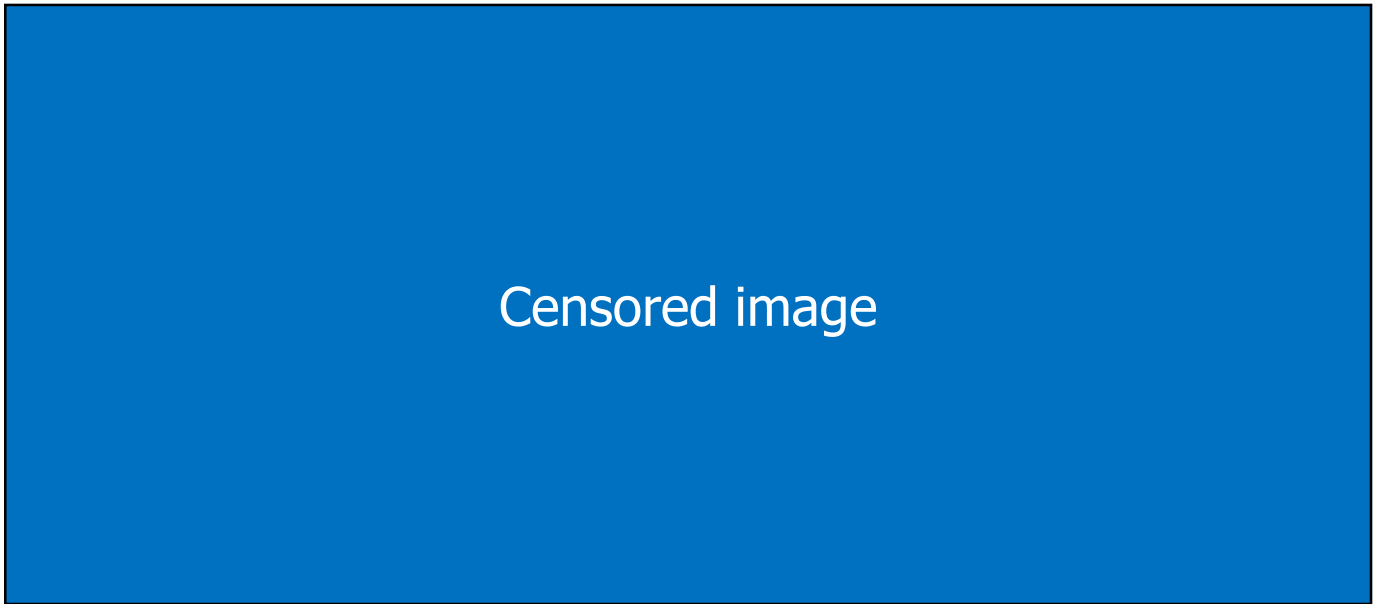
- Background
- Related work
- Proposed framework
  - Concept
  - Plug-in system
  - Scripting language
- Demonstrations
- Discussion
- <u>Conclusions and future work</u>

Conclusions

- We proposed a framework that makes it easy to create robot programs
  - No need to know the whole system
  - No need to compile and reboot
- Our framework is very useful in the RoboCup Soccer competitions
- Coming soon at following URL
  - http://www.shino.ecei.tohoku.ac.jp/jollypochie/

Future work

- Create a framework for robots other than AIBO, especially humanoids!
  - We hope that we can use some modules from AIBO, such as recognition modules



Thank you for your attention!

Censored image