UbiREAL Introduction

(1) Introduction to UbiREAL

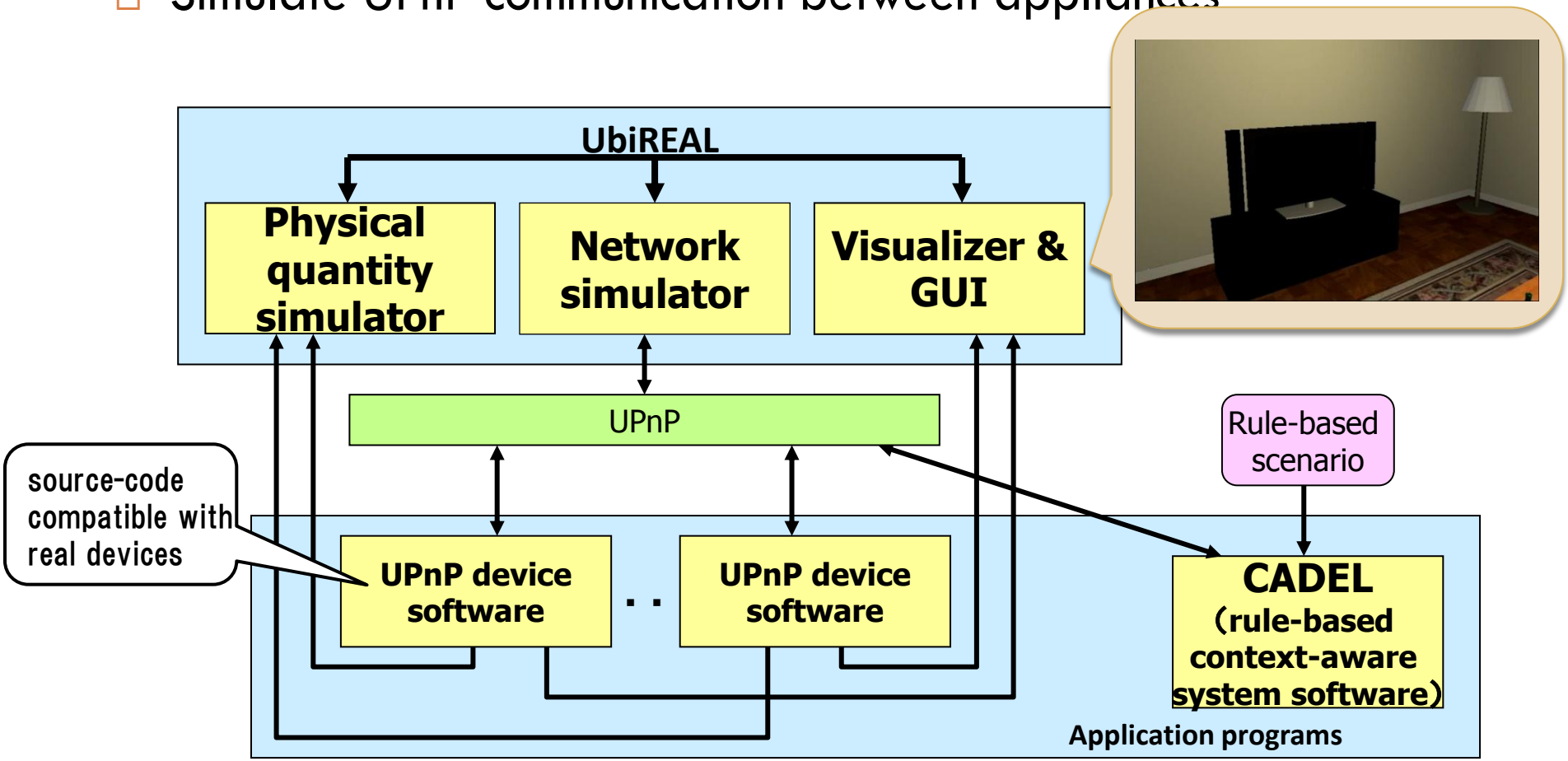
- Characteristics
  - High-speed graphics, short start-up time
  - Run on Windows, Mac (Intel CPU), Linux
  - Plugin function
    - Easy to incorporate new devices
    - Can newly define the underlying space
    - Can add new user mobility models
  - Can save configuration changed

Installing UbiREAL

- Installation
  1. Install JDK6 (32bit version)
  2. Unpack “ubireal-release-v1.0.zip” in your favorite folder
  3. Download clink170_ns.jar (http://www.ubireal.org/clink_ns/index.html), put clink170b_ns.jar in ¥ubireal-release-v1.0¥bin¥
  4. Download jogl-1.1.1-***.zip (http://download.java.net/media/jogl/builds/archive/jsr-231-1.1.1/) and unpack it (Select the ***.zip which is proper to your operating system)
     - Put jogl.jar and gluegen-rt.jar in ¥ubireal-release-v1.0¥bin¥
     - Put .dll(Windows)/.so/(Linux).jnilib(Mac) files in ¥ubireal-release-v1.0¥lib¥
- Execution
  - Windows: run ns.bat, cadel.bat, ubireal.bat, in this order
  - Mac/Linux: run run ns.sh, cadel.sh, ubireal.sh, in this order
- Folder organization
  - Root (.) : batch files
  - bin: binary files (*.jar)
  - lib: OS dependent files (*.dll, *.jnlib, *.so)
  - data: configuration files of UbiREAL (*.ur2) and Cadel rules (*.rule)

UbiREAL Structure

- Reproduce behavior of appliances (TV, air conditioner, etc)
- Simulate UPnP communication between appliances



CADEL: Example of context-aware control system

- Why we need CADEL?
  - UbiREAL provides only a virtual environment with context and network among sensors/devices
  - Application is required to control devices according to context change
- CADEL objective
  - Context-aware control of appliances
  - Easy and intuitive description of control scenarios
  - Avoidance of inconsistency in a scenario (i.e., conflict among rules)

How to describe scenario in CADEL

- Scenario
  - Describes context-aware controls as a set of rules
- Rule = (condition, action)
  - Condition: inequality for sensor values
  - Action: command for an device (appliance)

```
if(aliceBedroom1.Existence &&  sensorBedroom1.Temperature>=28){
   fanBedroom1.SetPower(true);
 }
```

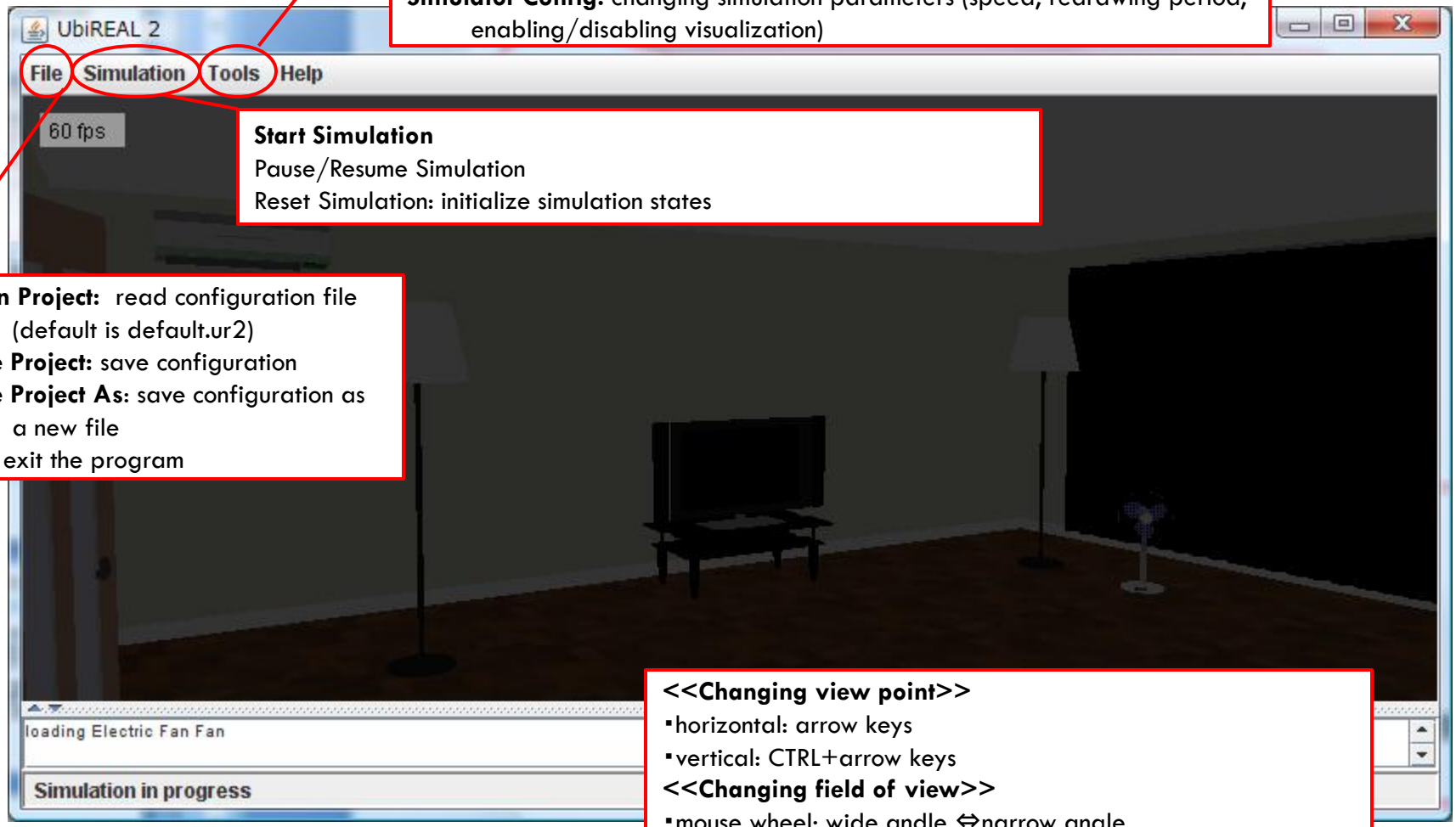
←condition

←action

- Another example of action: acBedroom1.SetTemperature(25)
- Names of devices and sensors are decided when they are configures as UPnP devices
- When adding a new device, you can name it as you want

How to use UbiREAL

<<Functions>>
Plugin Config Panel: Managing Plugin modules explained in the next page
Camera Control Panel: not implemented yet
Simulator Config: changing simulation parameters (speed, redrawing period, enabling/disabling visualization)



Start Simulation
Pause/Resume Simulation
Reset Simulation: initialize simulation states

Open Project: read configuration file (default is default.ur2)
Save Project: save configuration
Save Project As: save configuration as a new file
Exit: exit the program

<<Changing view point>>
- horizontal: arrow keys
- vertical: CTRL+arrow keys

<<Changing field of view>>
- mouse wheel: wide andle ⇔narrow angle
- default setting is wide angle (there is a some distortion)

Plugin Config Panel

- UbiREAL enables to add new functions as plugin modules
- Following plugins are incorporated as default
  - Clock, Temperature, UPnP communication
  - Ground, Sky, Plan of the house
  - Appliances (Lump, Air Conditioner, Heater, Electronic Fan, TV)
    - You can add in virtual space new entities of above appliances by operating: Config→Add new entity
    - You can change position, direction, size of each entity by:
      - Config→Move
  - Human detection sensors (at living room, corridor, bed rooms, …)
  - Mobility model of inhabitants
    - SimpleHumanModel: can specify trajectory
    - ProgramableHumanModel: can program mobility model by yourself

How to specify trajectory of inhabitant

(1) Select: Tools→Plugin Config Panel

(2) Double-click “SimpleHumanModel”

- Already registered entities (initially, only Alice) are shown in a list

(3) To add/change trajectory of an entity

→ Select the entity, and select: Config→ Edit Trace

- Other operations:
  - Adding a new entity: SimpleHumanModel→Add
  - Changing graphics: select entity→Config→Edit Parameters

Importing a new graphics object/device

- **Through "importer" of Plugin Config Panel, arbitrary object (in obj format) and device**

(a) Importing graphics object in obj format

(1) Select "importer"→ click "Config" button→ click "Add New Entity"

(2) Input appropriate name (e.g., sofa) in dialog box of "Input entity name"

→ importer Entity – sofa entity is created

(3) Select the created entity → click "Config"→ Edit parameters

(4) Input file name of the object (e.g., obj/sofa2.obj[*1]) you want to imported just after "filename=" and click "Apply"

(b) Importing device

- In the above step (4), input the class name (e.g., MyDevice[*2]) you want to import just after "classname=" and click "Apply"

*1: File name must be input as a relative path from ./data (need extension)

*2: File name must be input as a relative path from ./data/class (no extension)

Development and import of a new UPnP device

(1) Developing a new UPnP device

- UPnP device driver: develop with UPnP library ClinkX
- Graphics: develop with graphics library "toward3D"

(2) Importing the new device in a virtual space of UbiREAL and confirming its behavior

- Use "importer" of UbiREAL
- Describe various rules in CADEL and run simulation

(2) Example of implementing/incorporating a new device

- What you need to create a new UPnP device

  Step1: coding UPnP driver part of the device

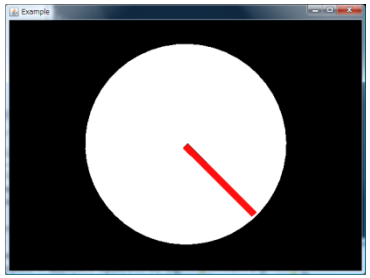
  → With UPnP library "ClinkX"

  Step2: coding 3D graphics part of the device

  → With graphics library "toward3D"

  Step3: coding simulator-dependent part

  Step4: incorporating the code into UbiREAL



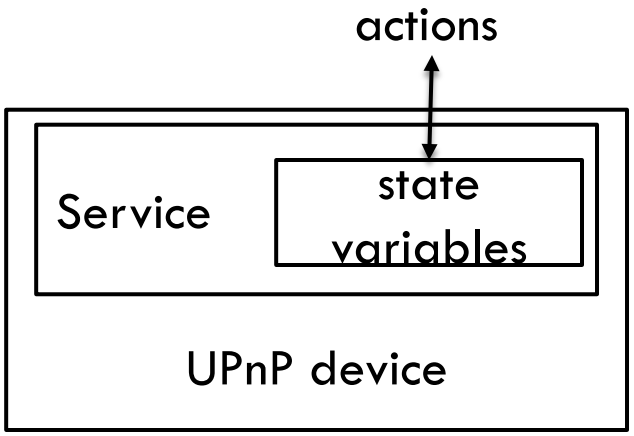
- Ex. Simplest clock with only a second hand

  Code: ./data/class/Example.java

  - Service: switching a state (on/off of power)
  - Graphics: cylinder + box
  - simulator-dependent part: second hand moves when power is on

Step1: making UPnP device driver part

- UPnP device provide several **services**
- Service
  - Consists of at least one actions and state variables
  - Ex. Turn on/off power, set channel to 2
- **publish/subscribe function**
  - When action happens (publish)→ subscribers are told



Template of UPnP device

- Simplest code without service

```
import org.itolab.morihit.clinkx.*;
import org.itolab.nsibata.ubireal2.plugin.Importable;
public class Example extends Importable implements UPnPActionListener {
  private final UPnPDevice device;
  public Example() {
    device = new UPnPDevice("Example");
    device.start();
  }
  public static void main(String[] args) {
    Example simple = new Example();
  }
}
```

You always need these

Create a new instance of UPnP device
Start the device (then, this is recognized as UPnP device)

This part is required if you want to test the program without UbiREAL.

Note: Here, file name is **Example**.java
In your program, you need to change the name "**Example**" to your favorite one

Adding a new service


```
import org.itolab.morihit.clinkx.*;
import org.itolab.nsibata.ubireal2.plugin.Importable;
public class Example extends Importable implements UPnPActionListener{
  private final UPnPBooleanStateVariable power;
  private final UPnPDevice device;
  public Example() {
    power = new UPnPBooleanStateVariable("Power");
    UPnPService service = new UPnPService();
    service.addStateVariable(power);
    power.publish();
    UPnPAction action = new UPnPSetBooleanAction(power.getName());
    action.setActionListener(this);
    service.addAction(action);
    device = new UPnPDevice("Example");
    device.addService(service);
    setPower(false);
    device.start();
  }
  public static void main(String[] args) {
    Example simple = new Example();
  }
  public void setPower(boolean p){
    power.setValue(p);
  }
}
```

Create boolean state variable "power"

Create a UPnP service "service"

Add "power" in "service"

Allow other devices to subscribe "power"

Create an action "action" (see next page)

Add "action" in "service"

Add "service" in "device"

Initialize "power" to be off

Implementation of "action"

You can define multiple actions by describing corresponding methods.

Note: be careful about the parameter types (see next page)

Parameter types used in action

- State variable, action, parameters of method should have the same type according to the following table

Type of state variable	Class of state variable	Class of action	Type of method that implements action
Boolean	UPnPBooleanStateVariable	UPnPSetBooleanAction	boolean
Integer	UPnPIntegerStateVariable	UPnPSetIntegerAction	int
Real	UPnPFloatStateVariable	UPnPSetFloatAction	float
String	UPnPStringStateVariable	UPnPSetStringAction	String

Step2: making 3D shape of device

- In this seminar, library **toward3D** is used
- Characteristics of toward3D
  - You can define arbitrary object by logical operation (and, or, sub) between multiple basic objects: box, cylinder, sphere, corn
  - Easy to parallely move, rotate, expand, reduce object
  - can specify scene graph, animation, camera position, light source
- --> For details, see http://ito-lab.naist.jp/~n-sibata/ubireal2/javadoc/

Step3: making simulator dependent part

- Structure of class "**Importable**"

```
public class Importable extends HLSceneGraphNode {
  public void initialize(UR2PluginIF m, UR2SimEntity e) {}
  public void registered(HLSceneGraphNode n, Vector3D pos) {}
  public void unregistered() {}
  public void simulationReset() {}
  public long simulationStarted() { return 0; }
  public long progressTime(long t) { return 0; }
  public long messageDelivered(long t, Message mes) { return 0; }
  public void updateAppearance() {}
}
```

  - **Initialize**: called only once when the entity is initialized
  - **registered/unregistered**: called when the entity is registered or removed
  - **sumulationReset/simulationStarted**: called when simulation started or was reset
  - **progreeTime**: called when simulation time reaches the specified value
  - **messageDelivered**: called when a message is received
  - **updateAppearance**: called before a graphics is drawn. You can specify changing shape here.

- You should override part of the above methods when your device works with simulator
- Also, see data/class/{Example.java, TemperatureMeter.java}

Step4: incorporation into UbiREAL

(1) Run UbiREAL (start_all.bat)

(2) Operate menu

- Tools→Plugin Config Panel→Importer→Config→Add new entity

(3) Input entity name

(4) Operate menu

- Select the added Entity→Config→Edit parameters

(5) In 2nd line: classname=, input the name of class you implemented

Note: save your java classes in directory "examination"

The above procedure registers the class in the virtual space

(6) You can change position, direction, and size of the object by:

- Select the entity→Config→Move

How to compile and execute

- Compilation and Execution
  - Open DOS window
  - Compile: jc.bat file_name.java
  - Execution: j.bat file_name
- Confirmation method
  - Run “ctrlp.bat”, then confirm if your device appears in UControlPoint window
  - Change the values of device states to confirm the device behavior
  - For initialization of the device, you need to execute “start simulation” in UbiREAL