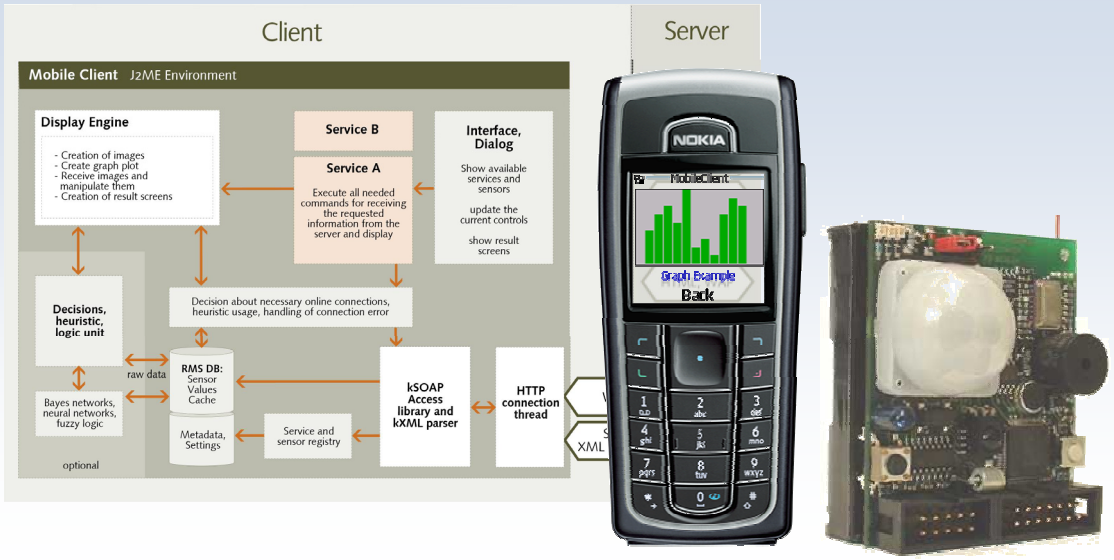
Bauhaus University Weimar



Final Presentation
Architecture, Components

Sens-ation

February 2005
Nicolai Marquardt
CML – Cooperative Media Lab
Prof. Tom Gross, Tareg Egla
Bauhaus University Weimar

Outline

1. Introduction
2. Sensor Adapter, Embedded Sensor Board
3. Sens-ation Server Classes
4. Client Applications

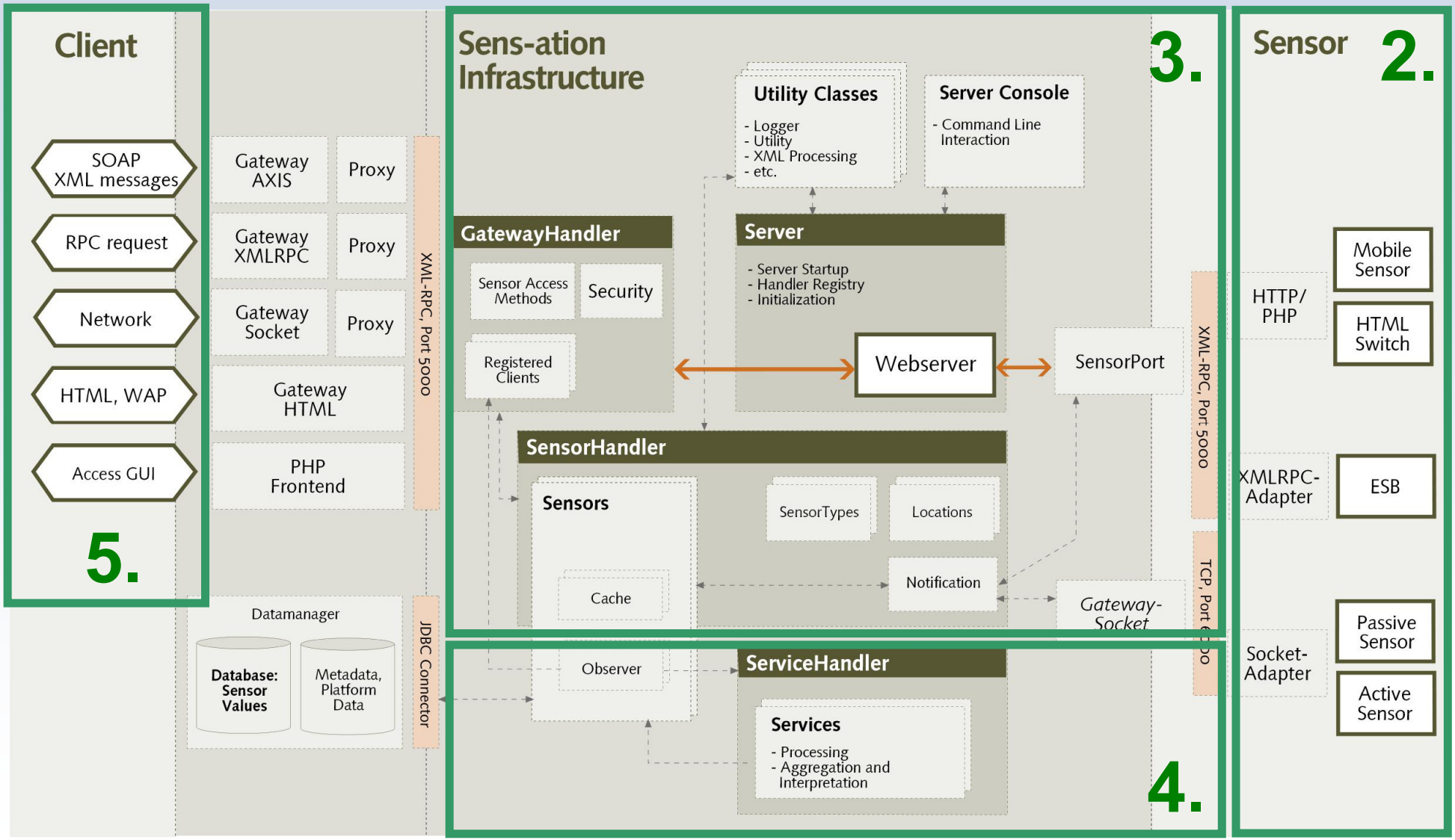
1. Introduction

1. Introduction

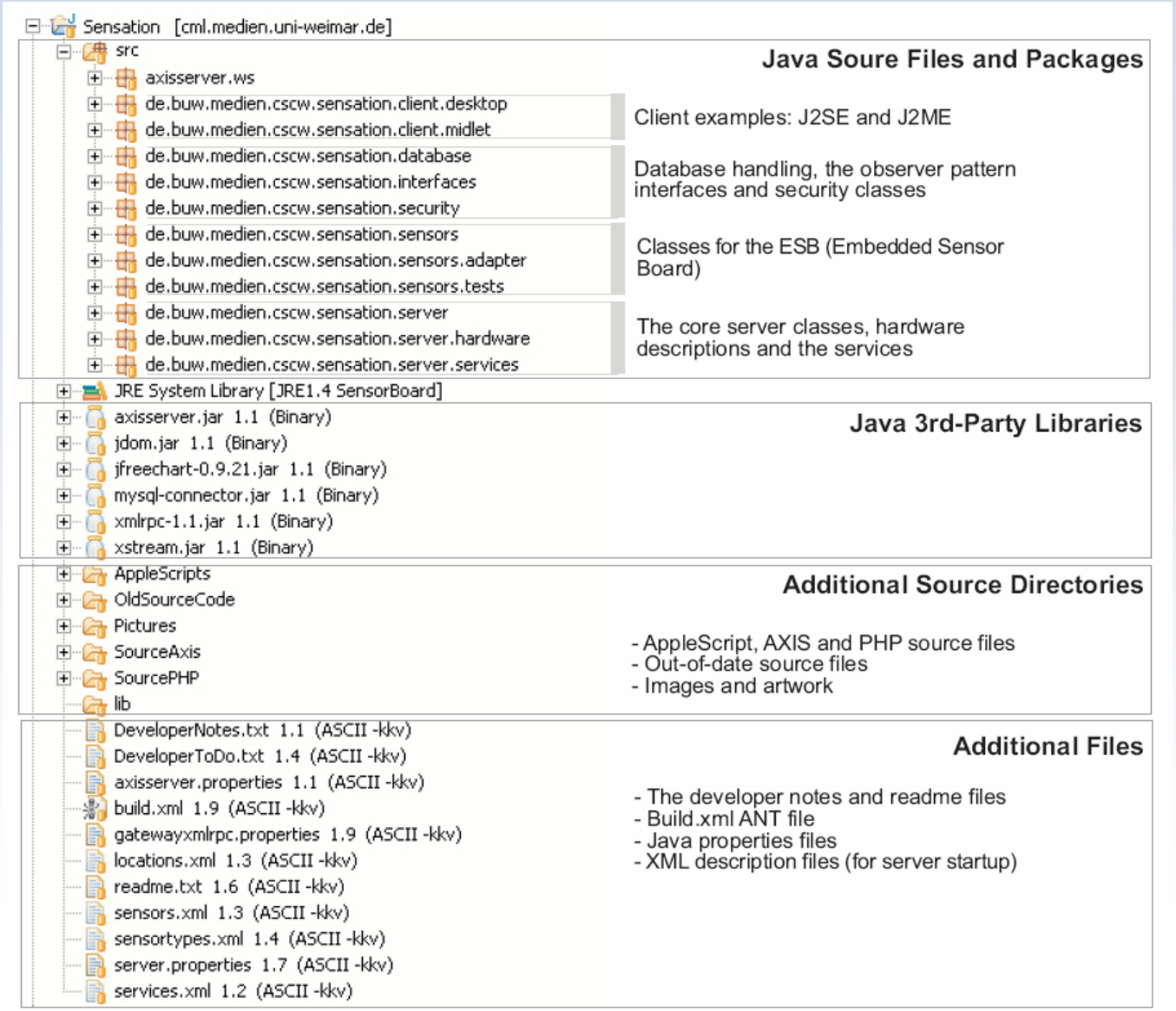
Features of the Sens-ation platform:

1. Easy integration of new sensors (providing many interfaces)
2. Various gateways for clients
3. Load balancing: core server, gateways and adapters can be located on remote computer systems
4. XML descriptions
5. Flexible exploration/discovery and access methods
6. Intelligent database
7. PHP admin interface
8. Service modules: enable interpretation, aggregation, filtering and calculations with sensor data
9. Example clients for J2SE, J2ME, AppleScript, HTML and mobile Portal

1. Introduction



1. Introduction



2. Sensor Adapter, Embedded Sensor Board

2. Sensor Adapter

Sensors of the Sens-ation Infrastructure:

1. **Hardware sensor module, e.g. ESB (Embedded Sensor Board) (wrapper for client access; adapter for communication to the server)**
2. Virtual sensor: keyboard input, mouse interaction, etc.
3. Mobile sensor: cell phone message, availability, profile, remote control
4. Instant messenger: send notification of the current presence state
5. Infrared remote control, WLAN connected devices
6. Hardware button and controls

2. Sensor Adapter

ESB – Embedded Sensor Board:

- Hardware module with integrated sensors
  - Temperature, movement (PIR), noise (microphone level, average, counter), light intensity, vibration, infrared RC5, hardware button
- Connected to the COM port, terminal commands
- Support of event pushing mechanism
- Wireless connections between sensor boards
- Infrared RC5 receiver and transmitter (→ Palm, PocketPC)
- Example response string:

```
[20|01.01.05 00:04:46|+026.0]
[IR:C(5) A(20)][Btn:0][Light:1046Hz][Pir:2]
[Vib:0][Mic:0][BAT:2266][EXT: 158]
```

2. Sensor Adapter

ESB – XMLRPC Adapter:

- Java wrapper for the communication to the sensor board
- XML-RPC connection to the server
- Process:
  1. Send commands and receive the value string
  2. Use parser to extract the sensor values (build value collection object)
  3. Build XML notification string of all values
  4. Send the XML notification via XML-RPC
- Parsing ESB messages, using regular expressions:

```
1  pattern = Pattern.compile("\\[Light:\\s[0-9]*\\sHz\\]");
2  matcher = pattern.matcher(toParse);
3  if (matcher.find()){
4    String light = matcher.group();
5    light = this.regularExpressionNumbers(light);
6    if (!light.equals("")){
7      collection.setLight(this.stringToInt(light));
8    }
9  }
```

2. Sensor Adapter

Adapter:

- Protect sensors from direct access of clients
- Caching methods of the adapter (and filter methods are possible)
- Implementations:
  - ESB board via XML-RPC and Sockets (TCP)
  - Java virtual sensor (keyboard): XML-RPC
  - Mobile cell phone sensors with HTML access
  - PRIMI with XML-RPC
- Active and passive sensors:
  - Active (Push): adapter is responsible for event notification (registration: polling interval, maximum time delay, etc.)
  - Passive (Pull): server can send request to the sensor (this is a sort of data collector) and is also used for accessing activators

3. Sens-ation Server Classes

3. Server

Main Classes (1):

- Server:
  - Main class of the Sens-ation infrastructure: Web server and handler registration
  - Initialization of the other server modules
  - Singleton (as well as: SensorHandler, ServiceHandler, GatewayHandler)
- SensorHandler:
  - Manages all sensors, locations, sensor types, hardware descriptions
  - Registration methods
  - Sensor and location discovery/exploration methods
  - Notification methods (to pass incoming notifications to the sensors)
  - Initialization: parses `sensors.xml`, `locations.xml` and `sensortypes.xml`
- Sensor:
  - Software side realization of "real" sensors
  - XML descriptions: `toXML` and `parseXML` methods, JDOM, SAXBuilder
  - Implements `subject` interface: clients and services as observers
  - Caching values
- SensorType:
  - Sensor classes: temperature, movement, noise, presence (PRIMI), etc.
  - Enable registration/access to groups of sensors

Sensor Infrastructure

XML Sensor Registration:

```
<Sensor id="MySensor" class="Temperature">
    <Description>Temperature sensor.</Description>
    <HardwareID></HardwareID>
    <Command></Command>
    <LocationID>B11</LocationID>
    <Owner>Nicolai Marquardt</Owner>
    <Comment>Sensor created with XML description</Comment>
    <AvailableSince>2005-01-01 11:54:37</AvailableSince>
    <AvailableUntil>2005-12-01 12:00:00</AvailableUntil>
    <SensorActivity type="active">
        <AverageInterval>10</AverageInterval>
        <MaxInterval>200</MaxInterval>
    </SensorActivity>
    <NativeDataType>Float</NativeDataType>
    <MinimumValue>-40</MinimumValue>
    <MaximumValue>50</MaximumValue>
</Sensor>
```

3. Server

Main Classes (2):

- Location:
  - Description of locations, with degree of longitude, latitude, etc.
  - XML descriptions: `toXML` and `parseXML` methods
- SensorValue:
  - For event notification: with sensor ID, date stamp and event message
  - Date and time: ISO 8601, yyyy-MM-dd hh:mm:ss.sss
  - Native type of entry: string, integer, float, xml
- XMLProcessing:
  - Parsing the XML collections: sensors, locations, etc. and create the objects
  - Handling the XML notifications: parsing and writing, pass to SensorHandler
- SensorPort:
  - Registered at XML-RPC handler (in server class)
  - Via XML descriptions creation of new sensors, locations (→ PHP, mobile,..)
  - Notification methods: split as parameter or send XML list of events
  - Update method for sensors

3. Server

Main Classes (3):

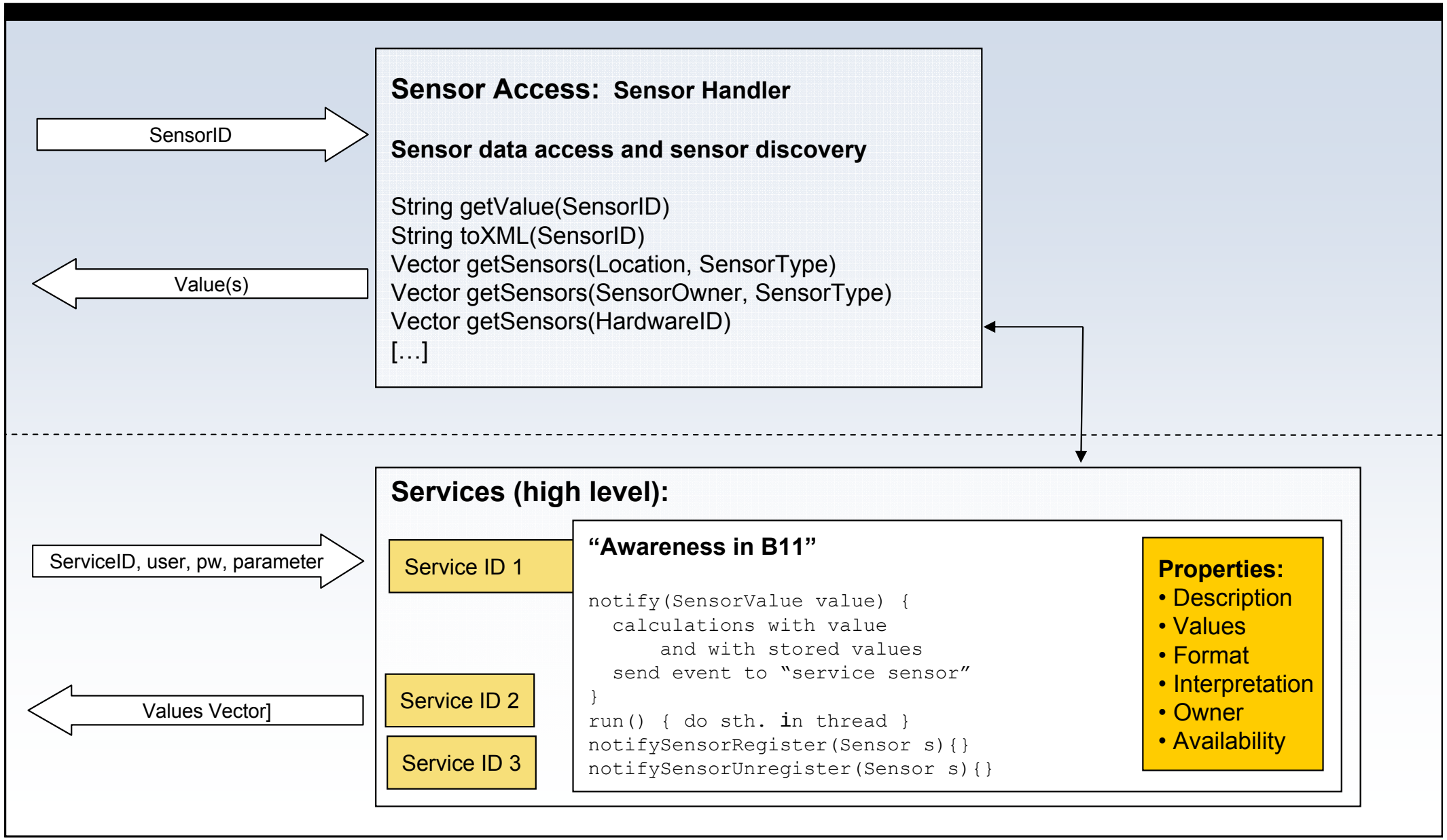
- ServerConsole:
  - Command line interaction with server module
  - Java Reflection API: class object of `Commands`, `getMethod` and `invoke`
- Commands:
  - Implementation of user commands for server interaction
  - List sensors, locations, delete, property, observers, clients, etc.
  - For each command separate method, parameter as vector (because of reflection)
- Preferences:
  - For access to the server configuration in `server.properties`
  - Methods for string parsing (conversions)
- ObserverInterface/SubjectInterface:
  - Java interface definitions, related to the observer pattern (Gamma et al.)
  - Notification of new sensor events

4. Services

4. Services

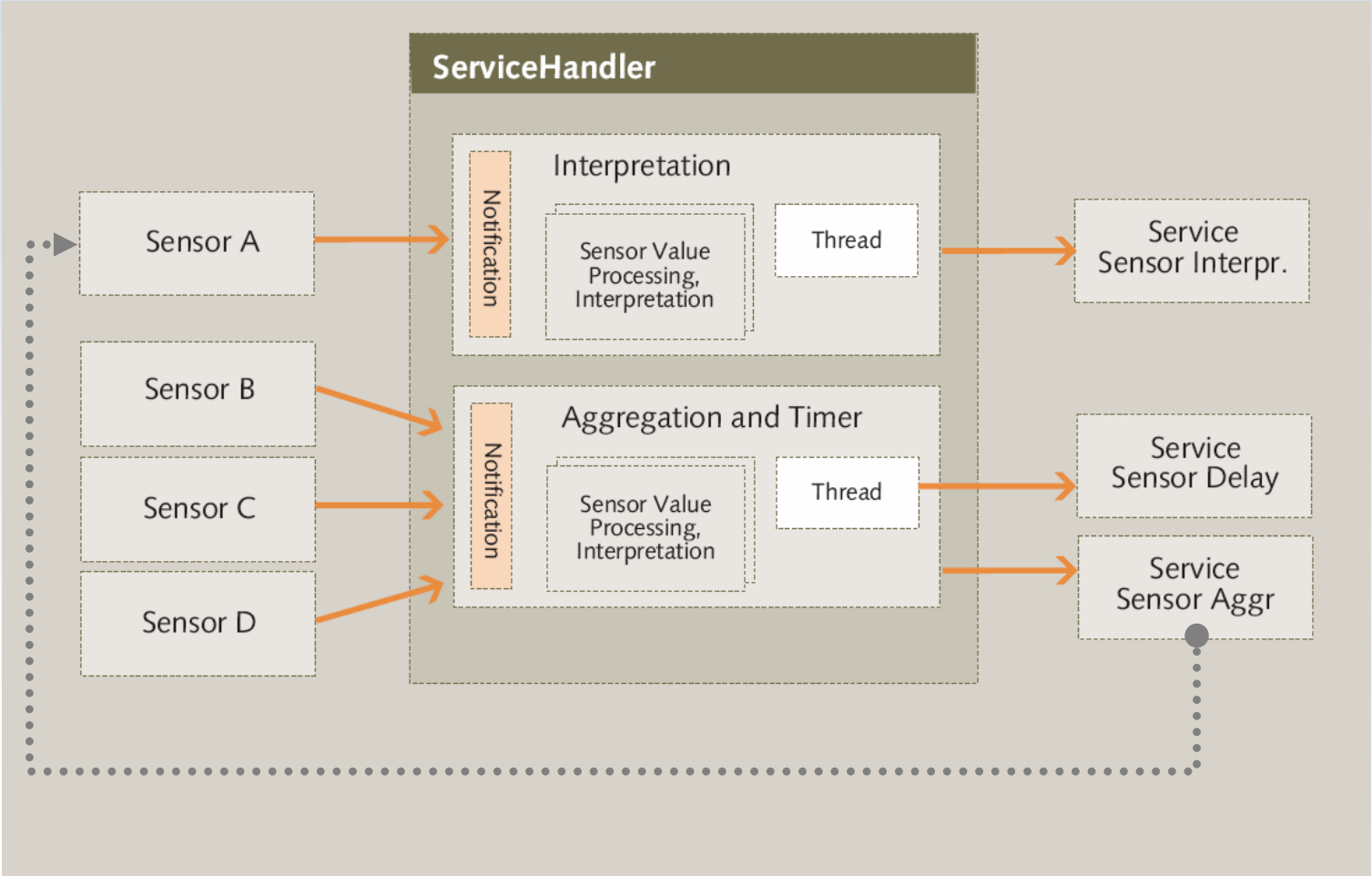
- Services:
  - Enable the interpretation of sensor data, or aggregation of various values as well as filtering, calculations, etc.
  - Extends the abstract service class (minimum interface methods and initialization)
  - Register own “service sensors” (at least one sensor) to publish results
  - Receive notifications of all sensors they are interested in
- ServiceHandler:
  - Handles all service objects (create, delete, etc.)
  - Provides exploration/discovery methods for services
  - Dynamic class loading of services: uses the URLClassLoader to load the class files of services specified in the `services.xml` file
  - This enables the easy extension of the platform with new services

4. Services

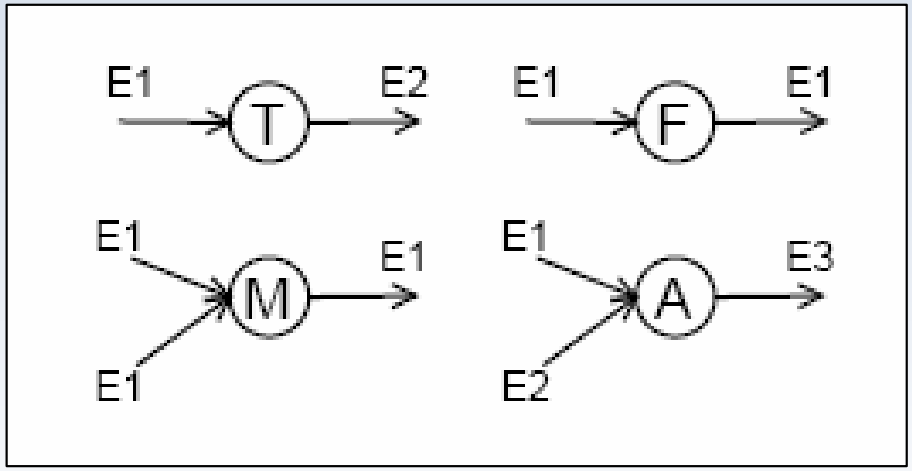

SensorID
Value(s)
Sensor Access: Sensor Handler
Sensor data access and sensor discovery
String getValue(SensorID)
String toXML(SensorID)
Vector getSensors(Location, SensorType)
Vector getSensors(SensorOwner, SensorType)
Vector getSensors(HardwareID)
[…]
Services (high level):
ServiceID, user, pw, parameter
Values Vector]
Service ID 1
Service ID 2
Service ID 3
"Awareness in B11"
notify(SensorValue value) {
calculations with value
and with stored values
send event to "service sensor"
}
run() { do sth. in thread }
notifySensorRegister(Sensor s){}
notifySensorUnregister(Sensor s){}
Properties:
• Description
• Values
• Format
• Interpretation
• Owner
• Availability

4. Services

Services:



4. Services



Reference: [Guanling Chen, David Kotz: Context Aggregation and Dissemination]

Why using services?

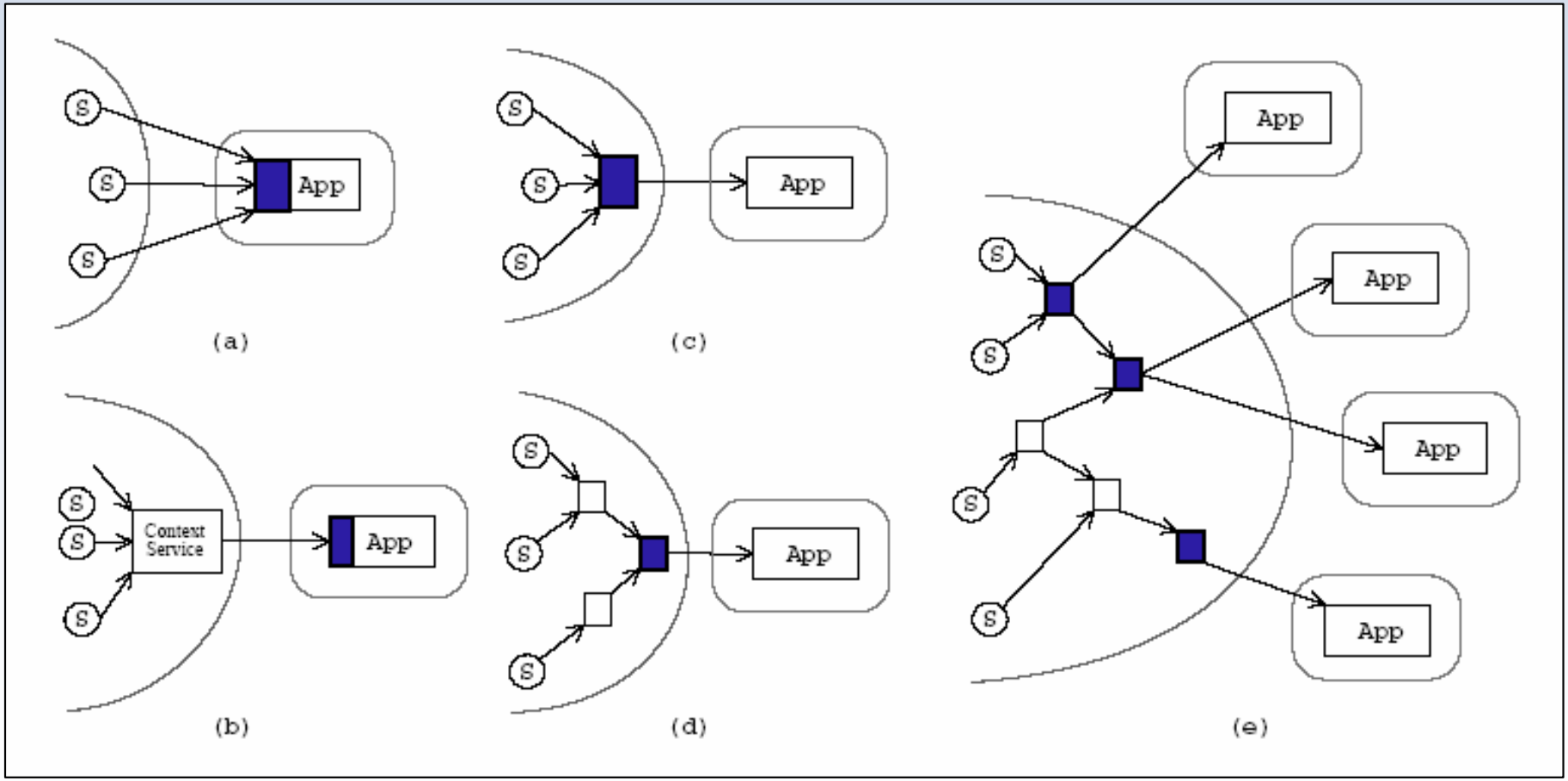
Types of operators [Chen, Kotz 2002]:

- Transformer (Interpreter)
- Filter
- Merger
- Aggregator

→ Building of operator graphs

→ Useful for heuristic calculations

4. Services



Reference: [Guanling Chen, David Kotz: Context Aggregation and Dissemination]

4. Services

Development of a New Service Module:

1. Extend the abstract service class, initialize `super("yourServiceID")`
2. Register your "service sensor": create the sensor object and register this sensor with `sensorHandler.addSensor(regSensor);`
3. Register the service class (`this`) as observer for all sensors you are interested in: discover by specifying sensor ID, location, sensor type, etc.
4. Implement the notify method: handle the incoming sensor events: **interpretations, aggregations, calculate the average value, filter, etc.**
5. Implement the run method of the thread (if needed)
6. Implement the `notifySensorRegister` and `notifySensorUnregister` methods (you can suspend the service If necessary)
7. Compile the services package (ANT file)
8. Add the class path to the `services.xml` file (for the dynamic class loader)

Example services in the `services` package: AwarenessService, InterpreterService, FilterService, MessengerService

5. Client Applications

5. Client Applications

AppleScript:

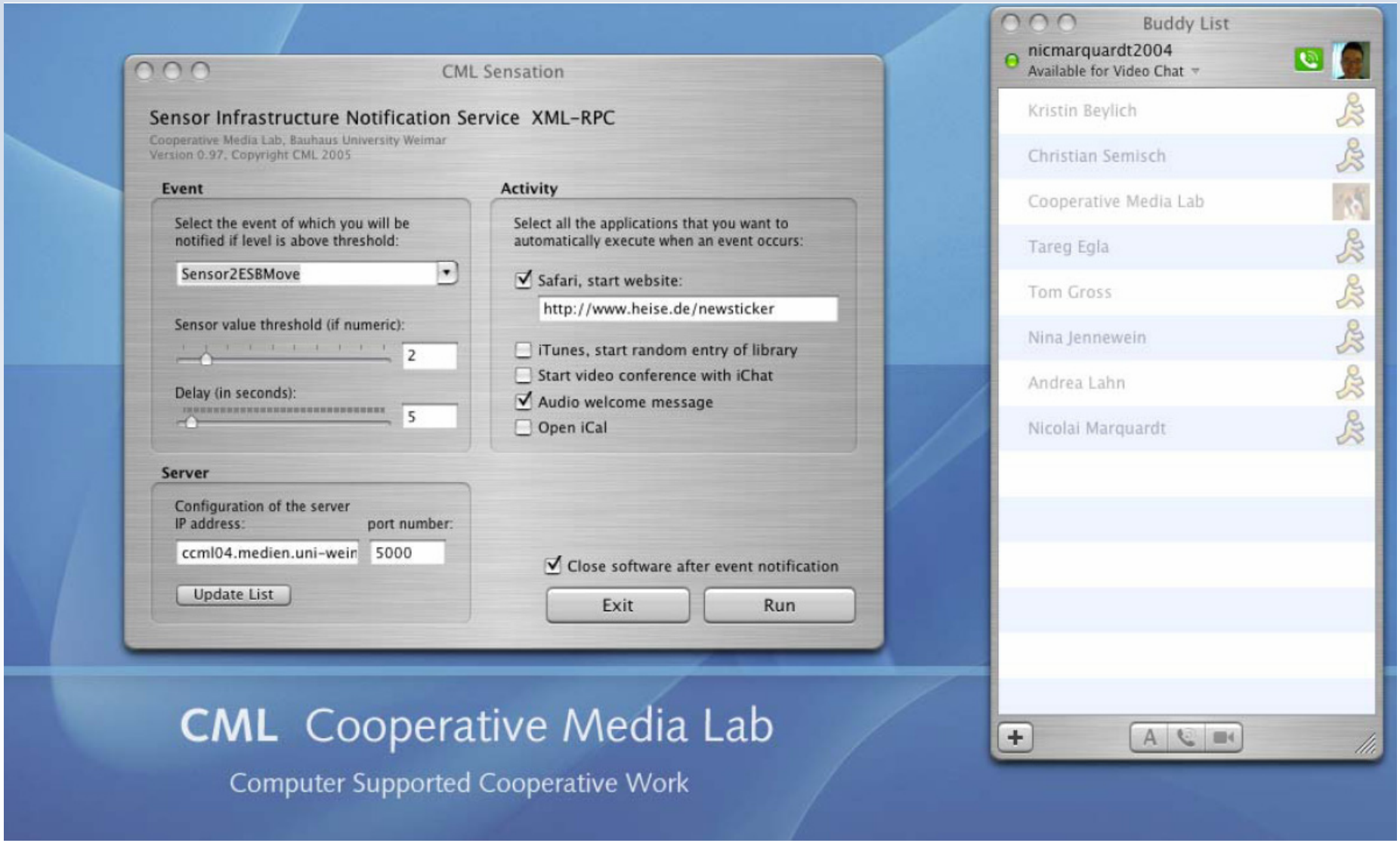
- Scripting language for Apple OS X (Aqua) operating system (similar to Visual Basic for Win32)
- Support for XML-RPC and SOAP calls
- Control of OS X applications: Safari web browser, iTunes (music player), iCal, iChat
- Development of a "AppleScript Notification Service":
  - Connects to server, updates sensor list
  - Select sensor and the application to control
  - The tool enables 'intelligent' reactions to the environment of a remote location
- User interface: using the Cocoa framework (and the Apple XCode IDE for development)

5. Client Applications

AppleScript: XML-RPC connection

```
1  on getValue(sensorID)
2    try
3      set method_parameters to sensorID
4      using terms from application "http://www.apple.com/placebo"
5        tell application "http://" & IP & ":" & Port
6          set this_result to call xmlrpc {method name:"GatewayXMLRPC.
               getValueString", parameters:method_parameters}
7        end tell
8      end using terms from
9      return {true, this_result}
10   on error error_message number error_number
11     set the error_message to "The script was unable to establish a
           connection"
12     return {false, error_message}
13   end try
14 end getValue
```

5. Client Applications

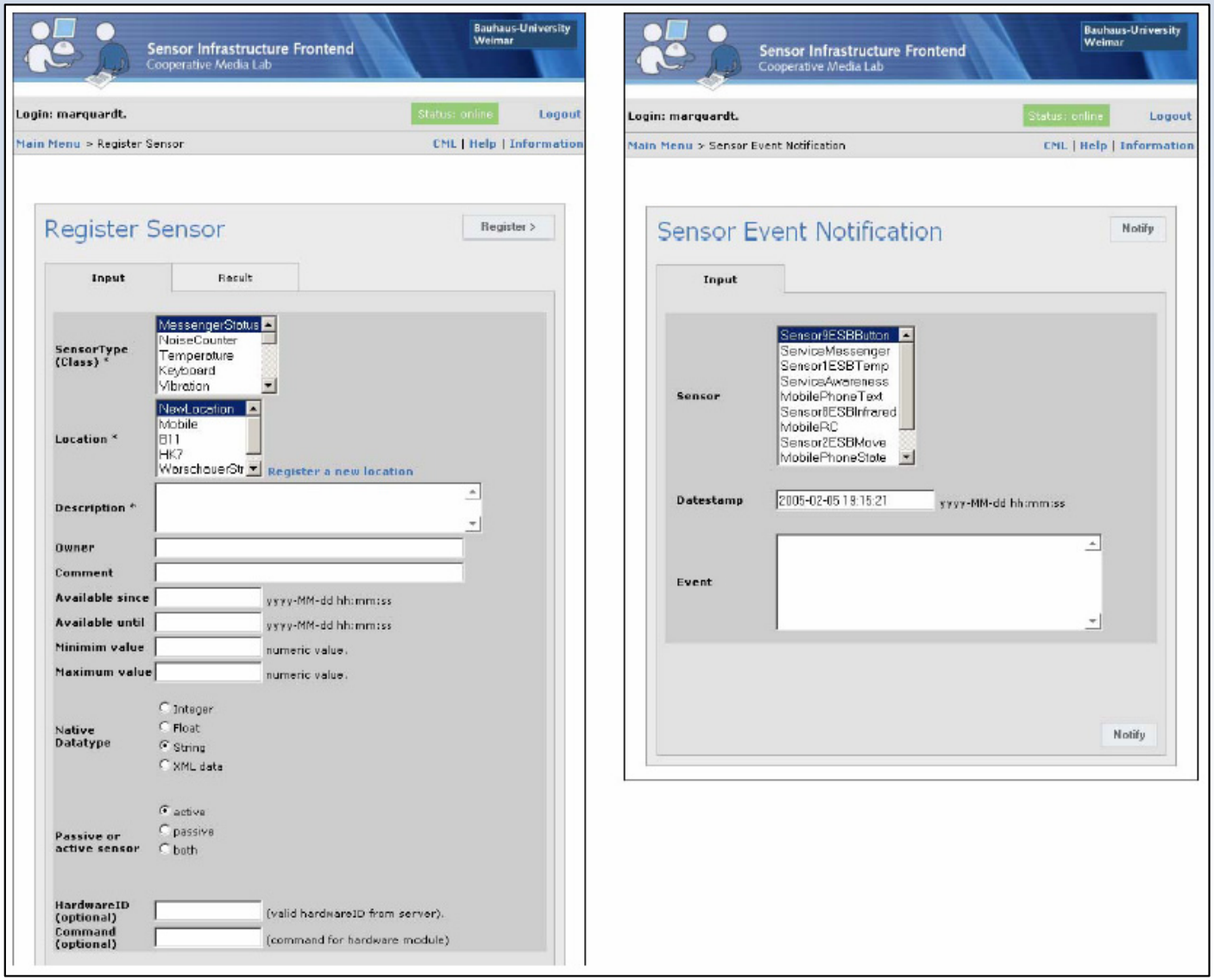


5. Client Applications

PHP Interface:

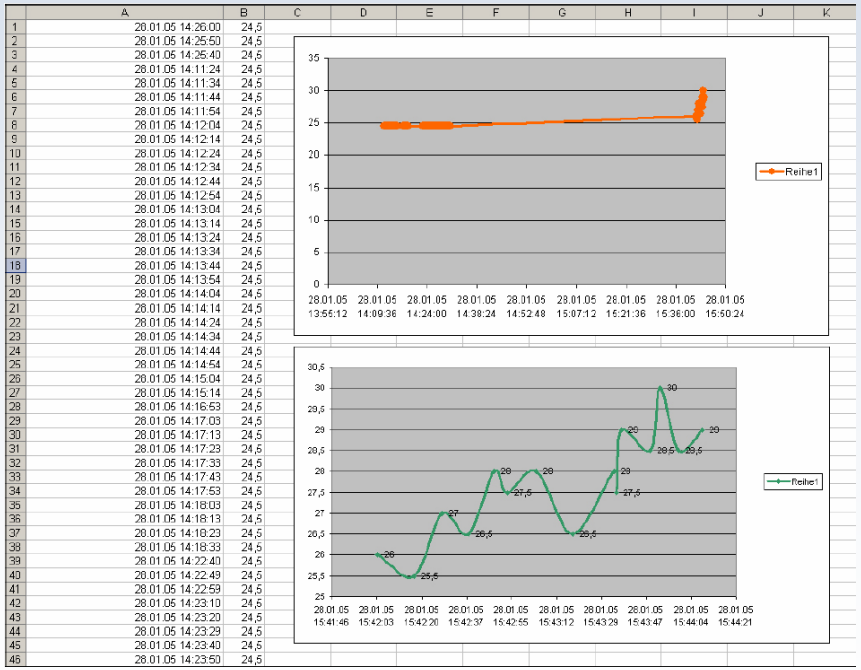
- Administration interface: register sensors, locations, publish events, access sensor values
- Visualization of graphs for sensor values and export in CSV
- Using the XML-RPC connection (with 3rd-party library)
- XML data creation and parsing
- PEAR graph engine for visualization
- Light-weight HTML gateway (adapter pattern)
- Mobile portal access: Access sensor values, display visualizations and send notifications interface for cell phone browser

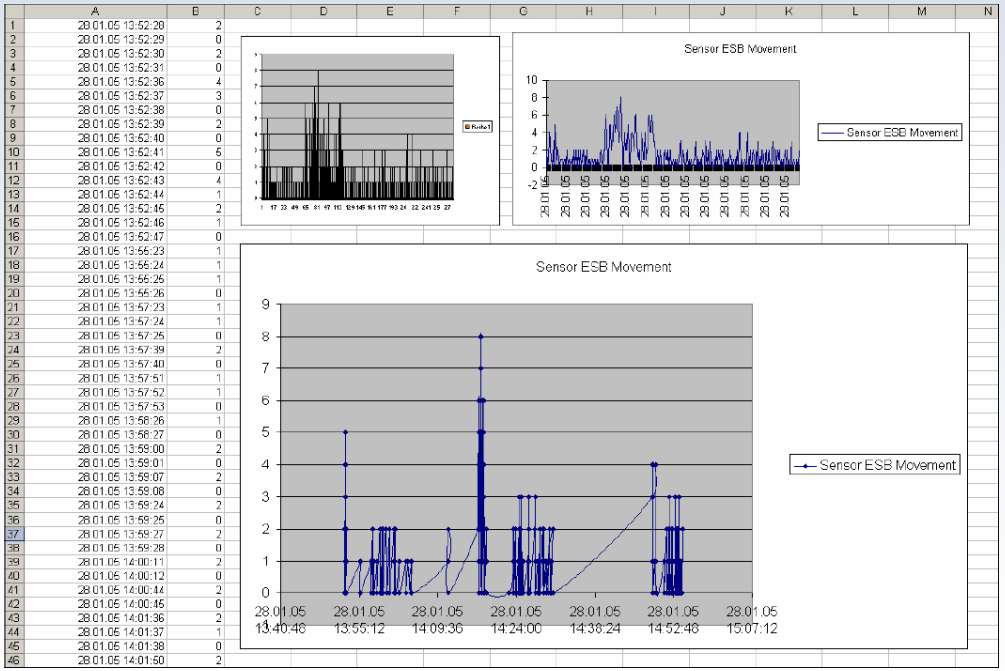
5. Client Applications



5. Client Applications

CSV data export:





5. Client Applications

Mobile HTML portal:

Thank You

For Your Attention!