UbiComp Lecture 4

eGadgets

Today these questions will be answered

- Are there ANY proposed technologies that support designers as well as people themselves, that enable them to create applications for Ubiquitous computing, without always have to start from scratch?
- And if there are, are they flexible enough, to possibly allow design with sustainability in mind?

Rationale

- To make an infrastructure that enables designers, but also people themselves, to make a wide range of applications with these new enhanced objects.

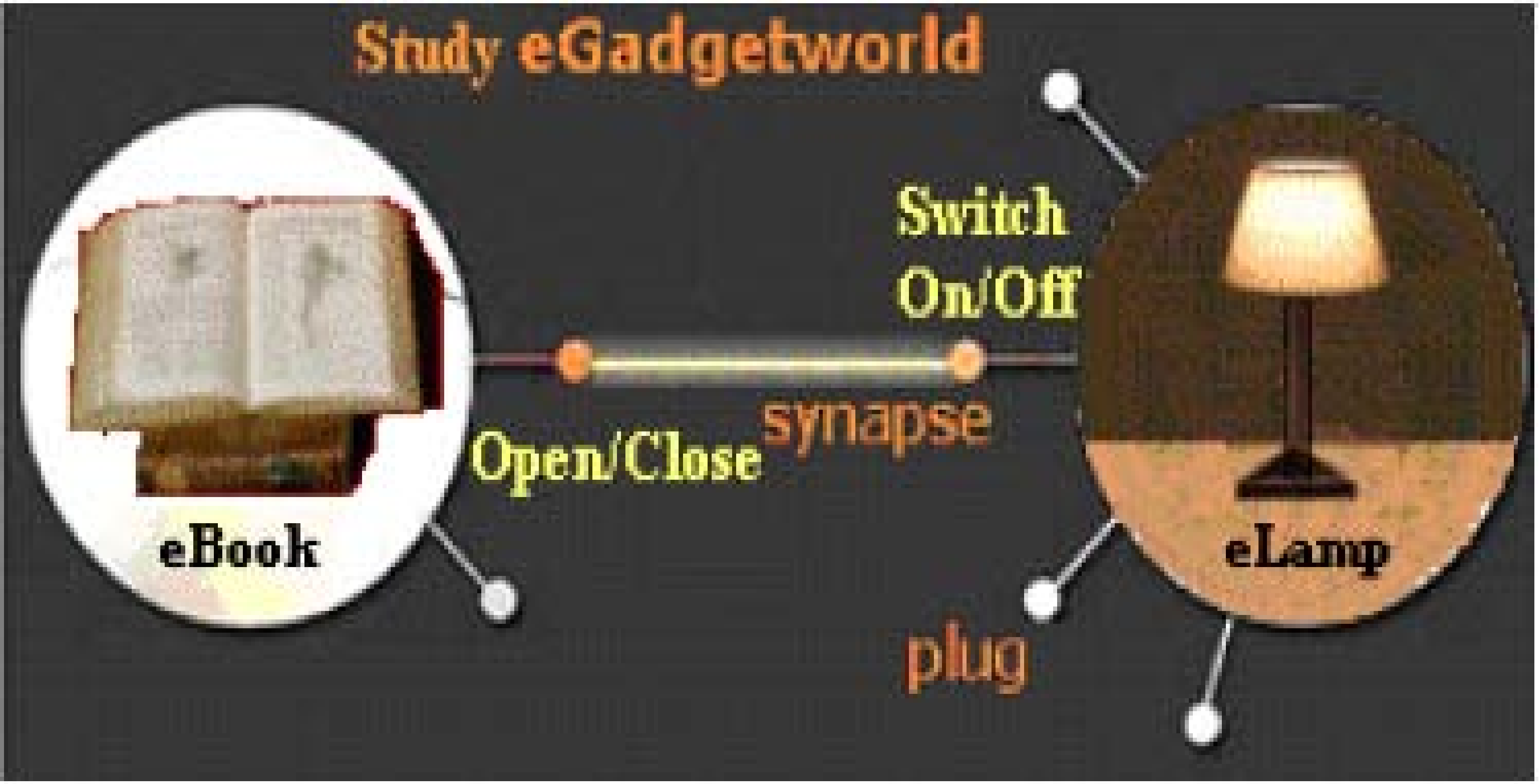
Paradigm for eGadgets

- People buy objects and make their surroundings from them, arranging them and rearranging them as fits their needs. They buy furniture, and stuff, from shops, but then they make up their own home landscape with them as they want, and rearrange it when they are bored with it, until they are satisfied.
- **We should be able to do this with our forthcoming digitally enhanced environments too**.

Innovation

- The overall innovation of the eGadgets approach is in viewing the **process** where **people configure and use complex collections of interacting artefacts**, as having much in common with the **process** where **software system builders design software systems out of software components**. In this approach we see the everyday environment as being populated with such artifacts, which people can associate in ad-hoc, dynamic ways. In this way new ***collective artefact behavior*** can emerge as a result of interactions among artifacts.

Example


Study eGadgetworld
Switch
On/Off
synapse
Open/Close
eBook
eLamp
plug

COMPUTING IN TANGIBLE

adapting the notion and principles of software architecture to the world of tangible artefacts

ITY ΙΝΣΤΙΤΟΥΤΟ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ
CTI COMPUTER TECHNOLOGY INSTITUTE

University of Essex

NMRC Ireland's ICT Research Centre

Basic Concepts

- **eGadget:** (http://www.extrovert-gadgets.net/) **Extrovert Gadgets (eGts) are everyday tangible (physical) objects that have communication abilities, and a range of sensing, acting, processing abilities.**
- Any gadget is of **dual nature**: it has

! a physical existence (demonstrated by its form and shape) and

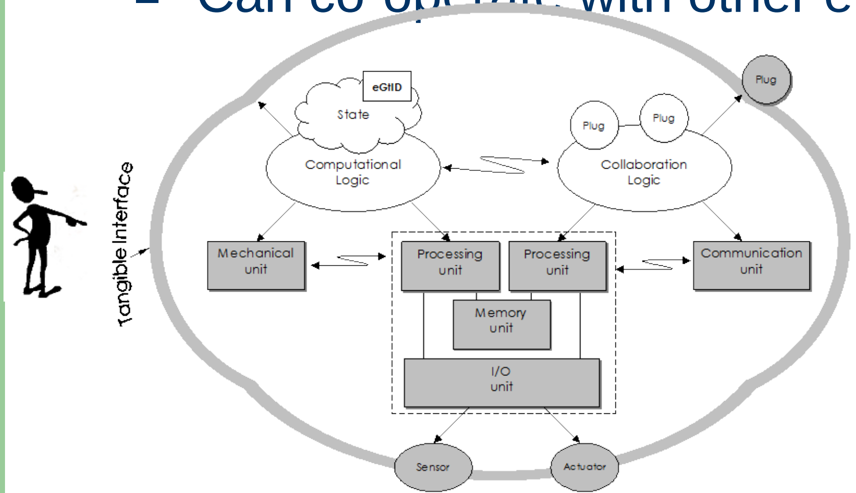
! an informational existence (implemented by a HW part and a set of SW modules)

Dual Presence of eGts

- In the real world they appear as tangible objects, occupying physical space for a certain amount of time.
- In the cyber (digital) world, eGts appear as software entities, which are instantiated and “run” on a processing unit.
- These two “selves” of an eGt are tightly interrelated: every eGt must have a representation in both worlds and changes in one representation may affect the other.

Let's talk about eGadgets

- An eGadget is an everyday tangible object, which:
  - Is able to process information and has an internal state
  - Has a set of abilities, offered as services and marketed through Plugs
  - Can communicate to other eGadgets at least its internal state and its abilities
  - Can co-operate with other eGadgets



An eGadget might also:

Be able to sense input from its environment

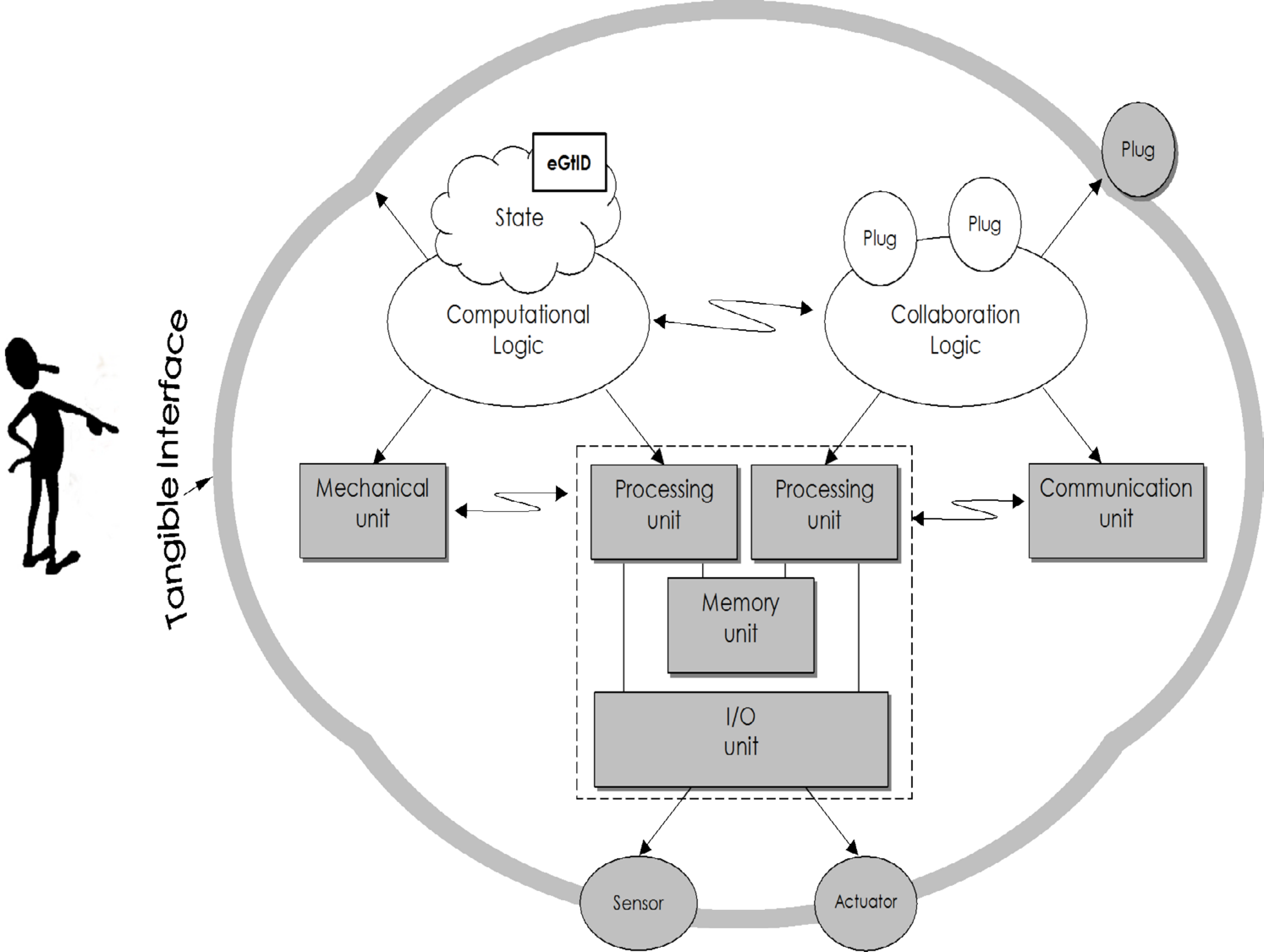
Affect its environment by action

Exhibit intelligent behavior

An eGadget has two selves, possibly detachable:

Tangible (tangible object, real-world properties)

Digital (software processes)

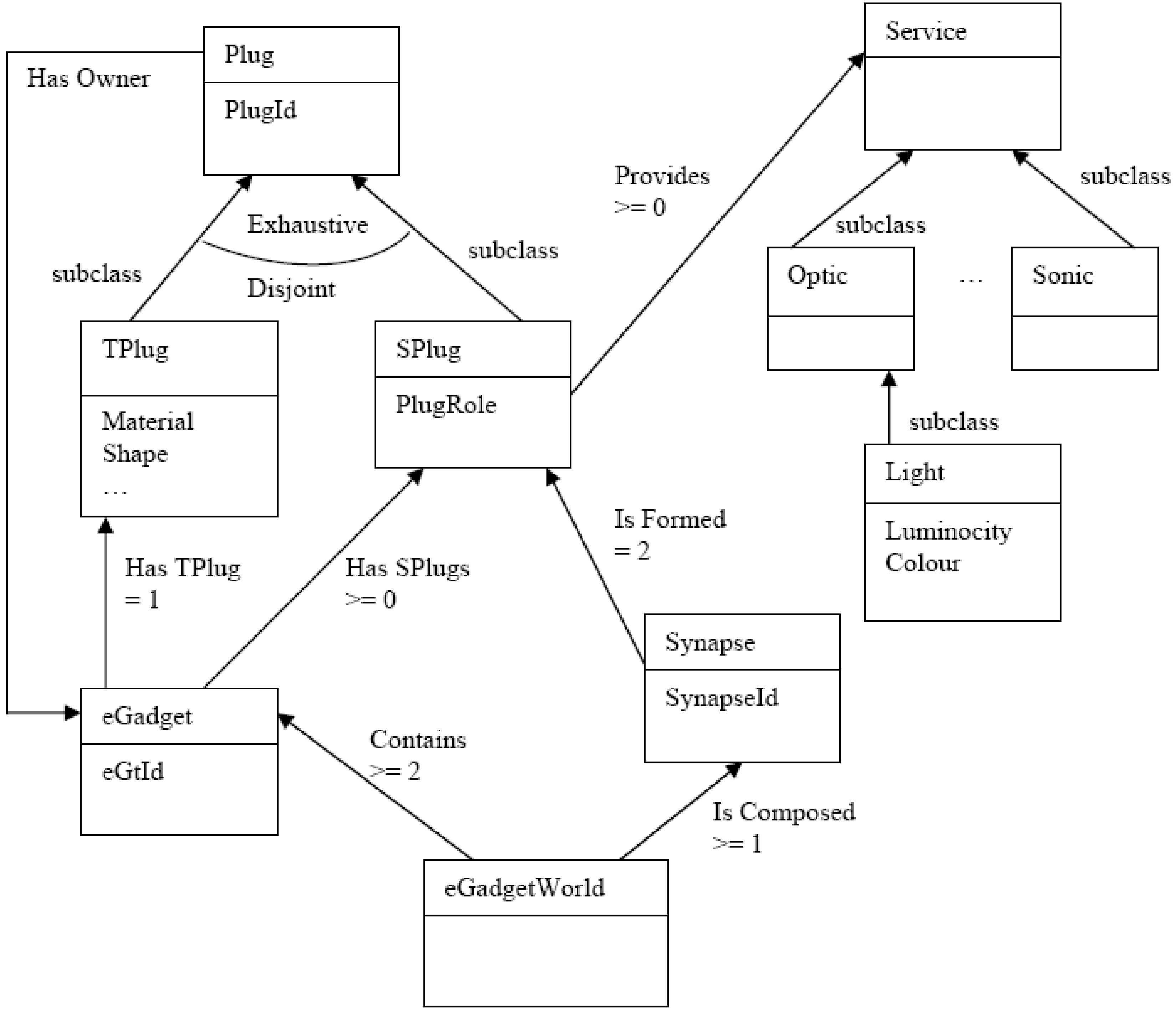

Tangible Interface
eGtID
State
Computational Logic
Plug
Plug
Plug
Collaboration Logic
Mechanical unit
Processing unit
Processing unit
Communication unit
Memory unit
I/O unit
Sensor
Actuator

Plugs

- Objects have several **capabilities**, coming out of their software or their tangible shelf. They have a physical shape, weight, colour, they may give services (lights for example), they may be squeezable, shakeable, grabable, liftable, containing, as you can understand this can go on for ever as it is a long list of cababilities.
- **Extrovert gadgets express their capabilities through plugs**. So other gadgets as well as people using them, know what to do with them (connect them, plug them to each other).
- **Capability Plugs are software classes** that make visible the eGt capabilities to people and to other eGts.

Plugs characteristics

- A Plug is an abstraction of the properties and abilities of an eGadget. It is implemented as a record and contains attributes and methods, which implement the ways it can be used (protocol), the service it can offer (methods) and its state (attributes). In fact, it is the only way other eGadgets can use eGadget services and have access to the eGadget properties.


Plug
PlugId
Has Owner
Exhaustive
Disjoint
subclass
subclass
TPlug
Material
Shape
…
SPlug
PlugRole
Provides
>= 0
Service
subclass
subclass
Optic
…
Sonic
subclass
Light
Luminocity
Colour
Has TPlug
= 1
Has SPlugs
>= 0
Is Formed
= 2
Synapse
SynapseId
eGadget
eGtId
Contains
>= 2
Is Composed
>= 1
eGadgetWorld

eGt

- Core term of GAS
- Represented as a class, which has a number of properties
- Physical properties - describe the eGt as a tangible object, e.g. shape, material and colour
- Digital properties, which manifest the digital self of the eGt, e.g. memory configuration, processing capabilities and communication interface and the plugs that are owned by the eGt and expose its services

Plug

- Class divided in two disjoint subclasses:
- **TPlug** describes the **physical properties** of the object that is used as an eGt (i.e. dimensions, shape, colour, etc) and lists all the eGt's Plugs and Synapses; **One TPlug for each eGt**
- **SPlug** represents the eGt capabilities; **an eGt can have an arbitrary number of SPlugs**
- For each SPlug that participates in a synapse, a special role can be declared through GAS-CO.

Synapses

- Are associations between two compatible Plugs. They are invisible links, explicitly created to achieve a particular working of the two capabilities together.

- A Synapse may only appear among two SPlugs

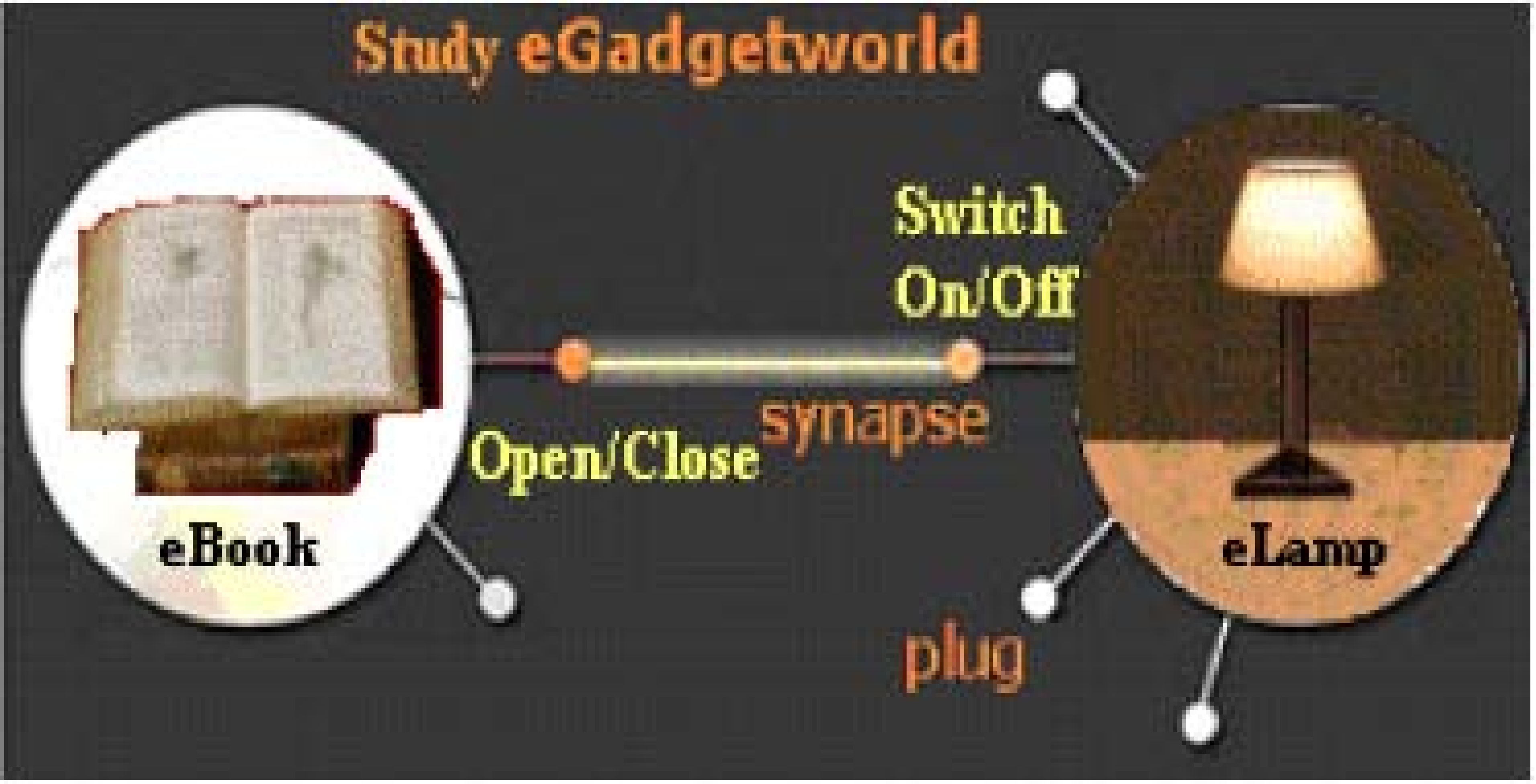
Service

- eGt through an Splug provides a number of services.
- [illegible] transmit/perceive.
- GAS-Ontology describes a service classification, which may be based on the type of the signals that an actuator/sensor transmits/perceives.
- The class service is divided into subclasses, each of which corresponds to one of the above elementary forms of signals (electric, electromagnetic, gravity, kinetic, optic, thermic and sonic)
- A service can be further refined into higher level services: e.g. the optic service can be refined into light, image, etc.
- Additionally a service may have a set of properties; e.g. light can have colour, luminosity, etc.

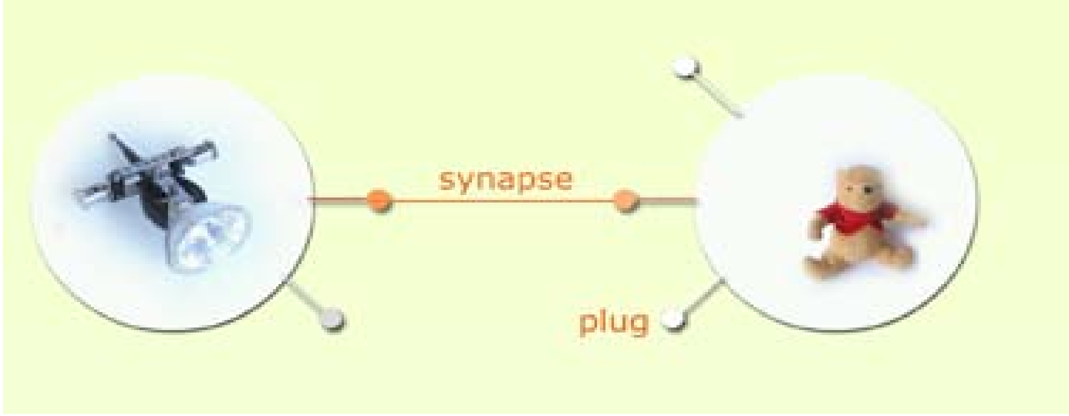
eGt Characteristics

- An eGt is autonomous, perceived as one entity, though it may have internal structure.
- It has an ID, a set of Plugs and an internal state, which it manages locally.
- It can participate in Synapses via its Plugs.
- It can always state its ID, set of Plugs (and their state) and active Synapses (and their state) in a universally understandable way at a predefined communication channel. Moreover, using a set of intelligent mechanisms, an eGt may be able to locally optimize or adapt its behavior, or even optimize the behavior of an entire eGadgetworld.

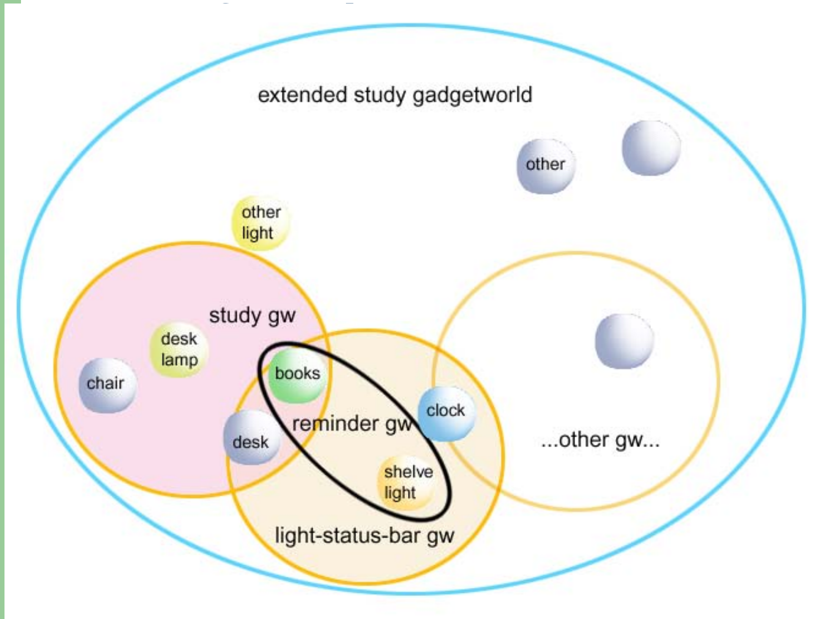
Example


Study eGadgetworld
Switch
On/Off
synapse
Open/Close
eBook
eLamp
plug

GadgetWorlds

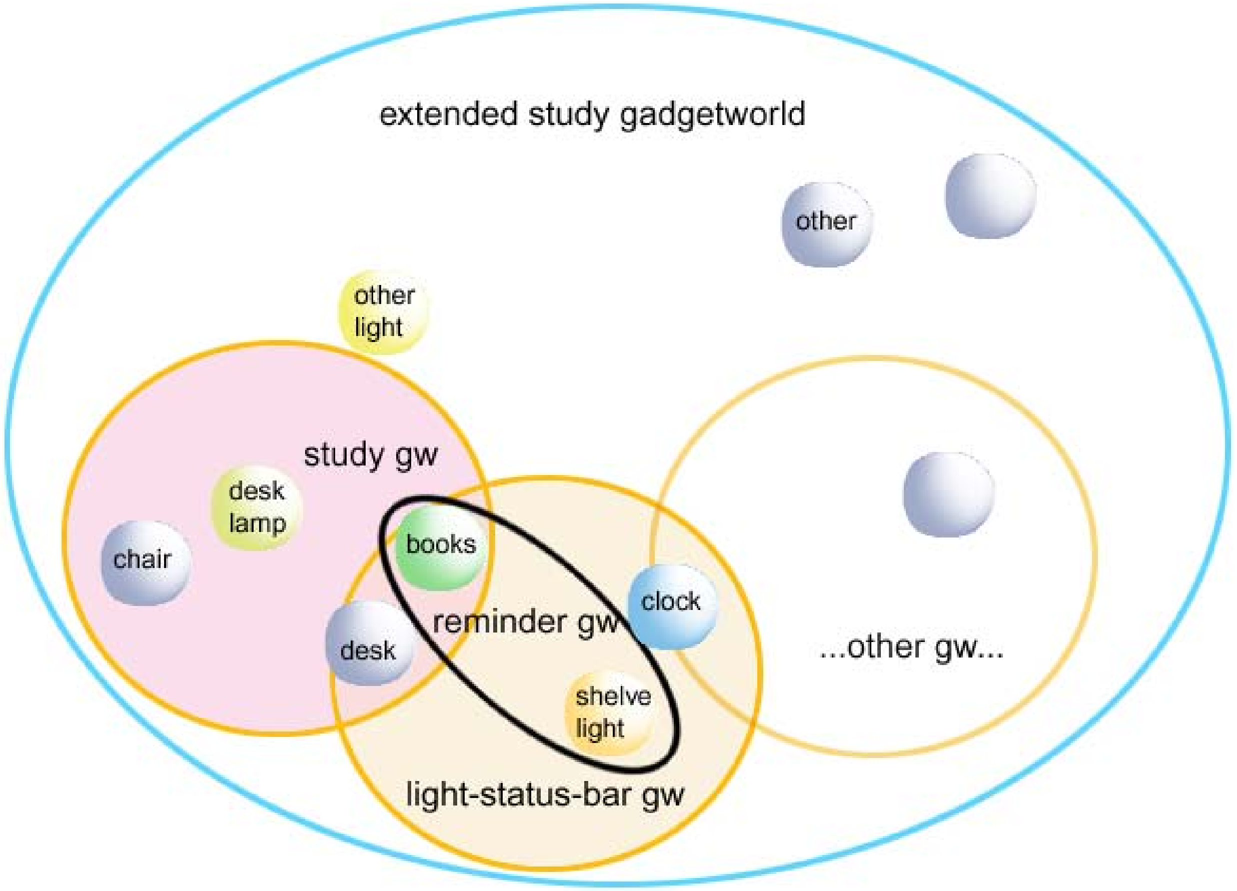


- A Gadgetworld is
  - **A collection of eGadgets, composed by people, which perform collective**



Once composed, people can use Gadgetworlds as identifiable entities and save / restore, edit / destroy, use or carry them.

About 10 different objects are now converted
A first prototype software tool, the Editor, helps to mar
synapse links between the



Exercise

- Work in pairs
- Think of a possible ‘eGadgetWorld’ you can form with the illustrated artefacts
- Give example of Plugs and Synapses that would created your eGadgetWorld
- In 10 min I will collect your work

GadgetWorld

- A distinguishable, specific configuration of associated eGts formed purposefully by a designer, a user, or even an intelligent agent.
- A configuration of gadgets which communicate and / or collaborate in order to display **a collective function**

! is formed by coupling gadget plugs

! we are interested mostly in the so-called "meaningful" gw, which are formed by a user to explicitly serve a purpose

! a gw is an open system

Are eGts about people?

Can be about any intelligent agents

Goal: I want to study

- People who possess eGadgets, each of which offers certain services, set a goal, which can be achieved as a collective eGadget function.

eGadgets I can use:
Desk, Chair, Lamp, Book

Synapses I must form:
Desk <-> Chair
Desk <-> Lamp
Desk <-> Book

They pick the eGadgets they need

They associate the eGadgets by establishing Synapses and form a Gadgetworld.

A Synapse is an established association between two Plugs.

A Plug is a communication port with a description of a certain service; a Plug is typed.

Approach

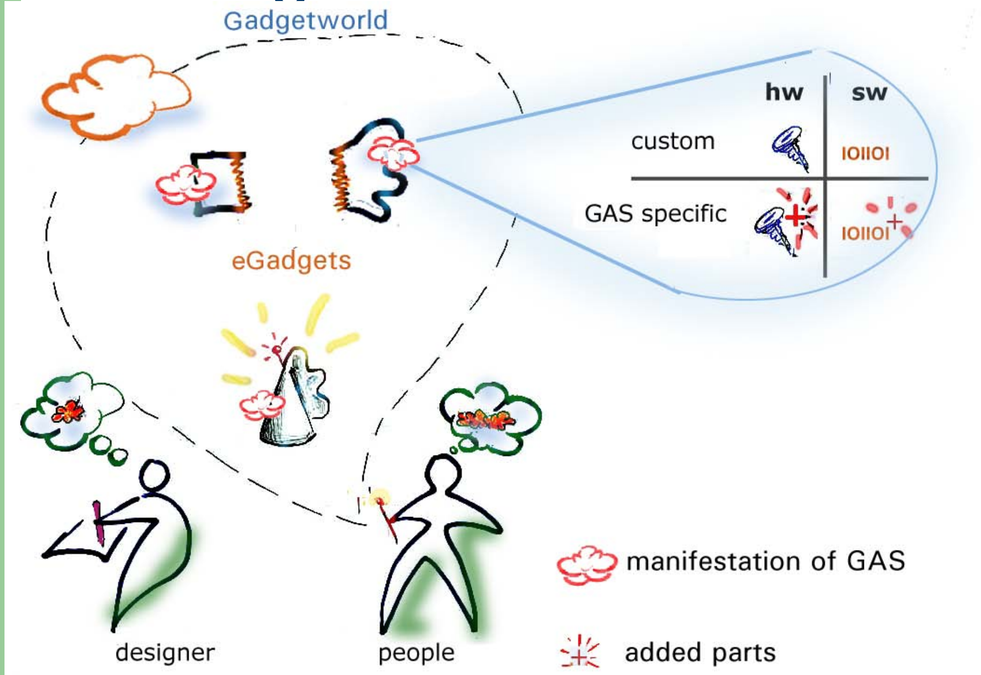
- Composition of eGts into an eGW is similar to the construction of a software application using software components.
- Thus, the “extrovert Gadgets” project approaches the problem both from software engineering and design engineering perspectives

Gadgetware Architectural Style (GAS)

- The generic framework that supports these add hoc connections, is called Gadgetware Architectural Style (GAS).
- This is a style that defines the concepts and mechanisms that allow people to create Gadgetworlds

GAS (Gadgetware Architectural Style)

- A conceptual and technological framework for describing and manipulating Gadgetworlds.



GAS consists of

- A vocabulary and layers of semantic associations between terms
- A set of configuration rules
- A technical infrastructure to support it

GAS can be an underlying technological infrastructure, for researchers that want to do home-automation, experiment with agent based intelligence, etc

Where is it?

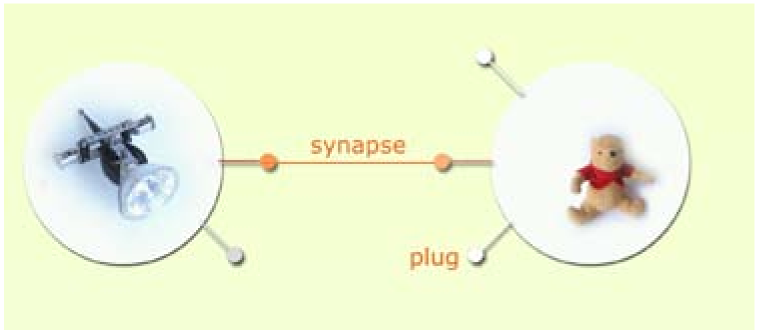
- GAS lies:
  - In the minds of designers of eGadgets
  - In the minds of people who compose Gadgetworlds
  - In the collaboration logic of eGadgets

GAS is independent:

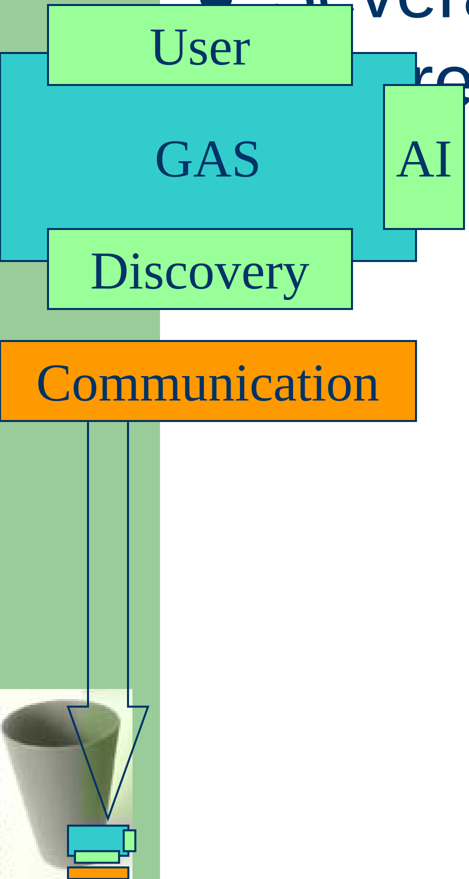
Of the internal working of the artifact

Of the communication protocol

Of the service discovery protocol



Issues to be addressed



- Several issues are being addressed at two rent levels:
  - eGadget level
  - Gadgetworld level

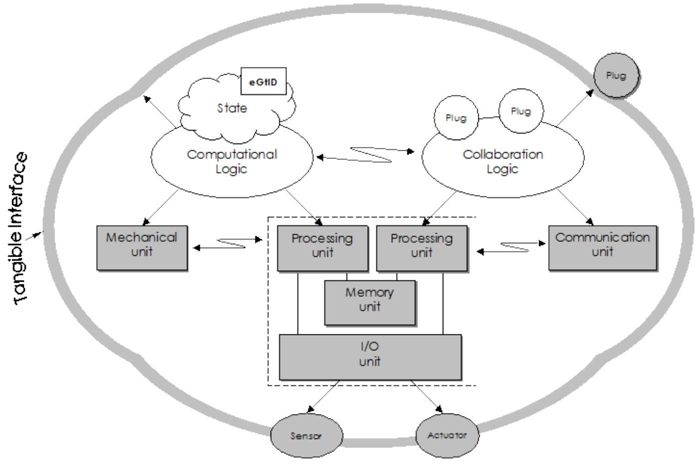
… and regarding different aspects:

- People usage of eGadgets and Gadgetworlds
- eGadget design
- Gadgetworld realization
- Role of intelligence
- Integration, packaging and miniaturization
- Dependence on available and projected technologies

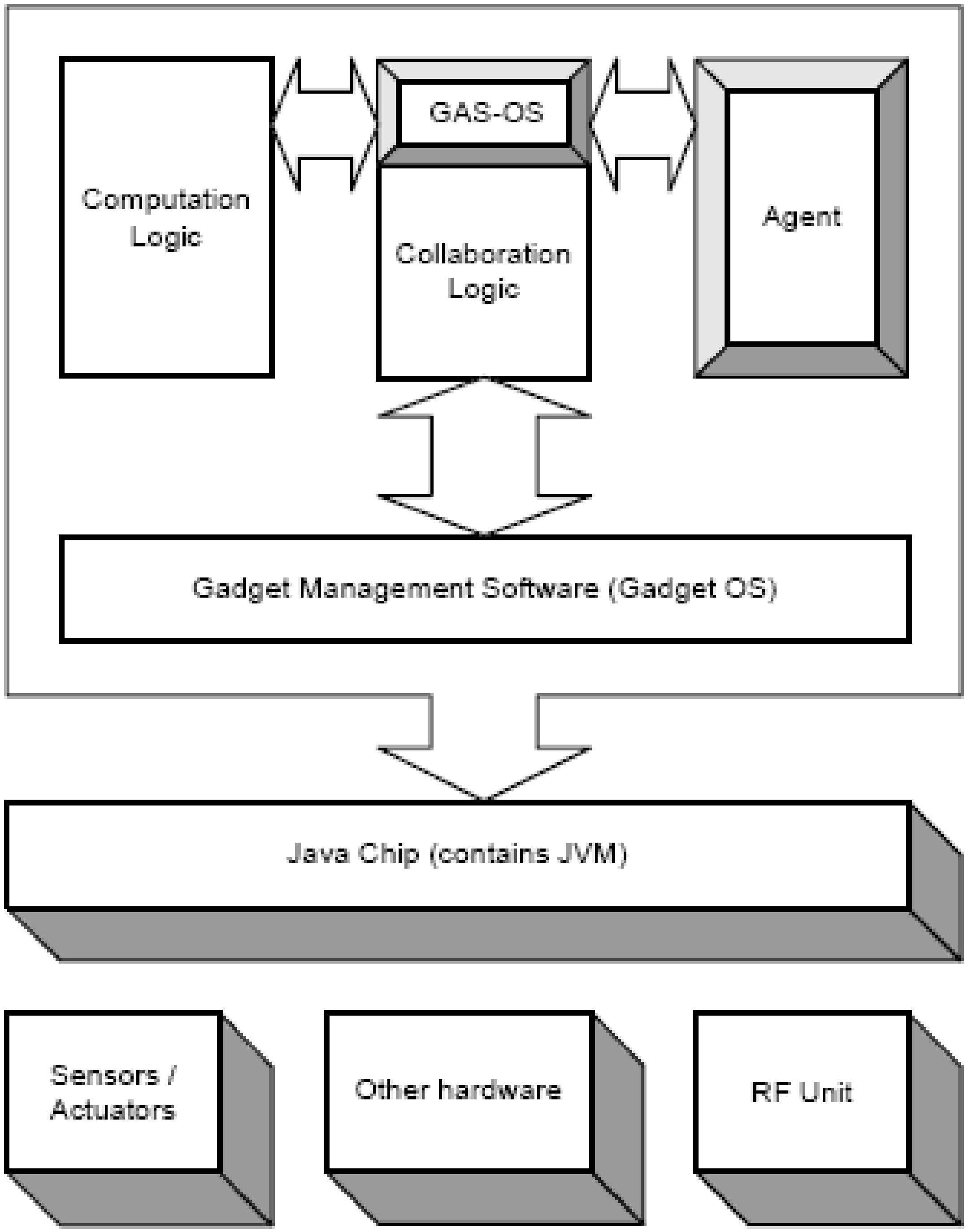
GAS Consists of:

- A set of architecture descriptions (syntactic domain)
- A set of guidelines for their interpretation (semantic domain)
- A set of mappings from the syntactic domain to the semantic domain
- A set of constraints or rules that represent the application domain

GAS

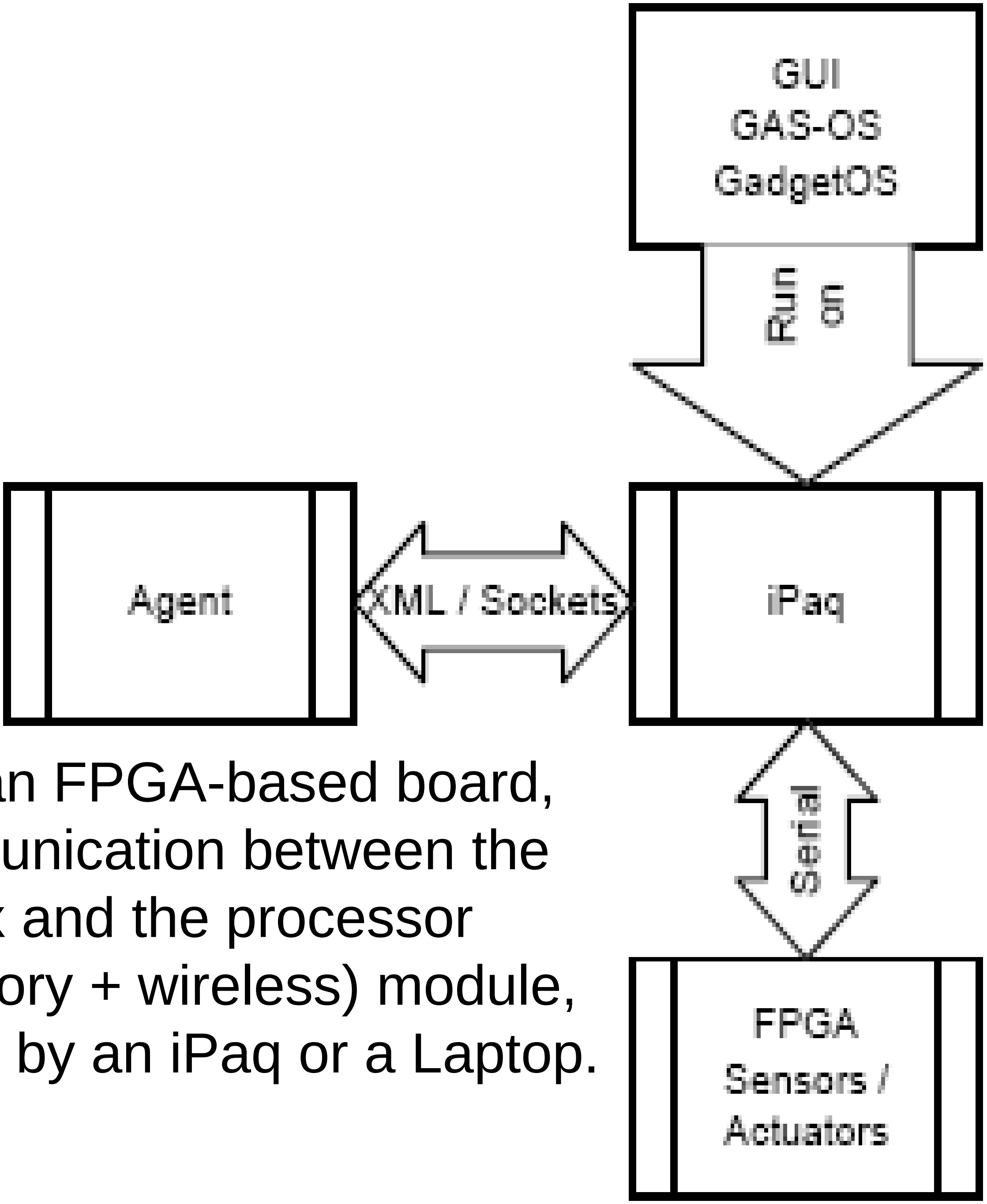


eGtID
State
Plug
Plug
Plug
Computational Logic
Collaboration Logic
Tangible Interface
Mechanical unit
Processing unit
Processing unit
Communication unit
Memory unit
I/O unit
Sensor
Actuator



Computation Logic
GAS-OS
Collaboration Logic
Agent
Gadget Management Software (Gadget OS)
Java Chip (contains JVM)
Sensors / Actuators
Other hardware
RF Unit

Detailed architecture



An eGadget is made of an FPGA-based board, which implements communication between the sensor or actuator matrix and the processor and a (processor + memory + wireless) module, which is currently served by an iPaq or a Laptop.

GAS Middleware

- GadgetOS module – specially implemented per every eGt and used to control its resources
- GAS-OS module - manages the Plugs and Synapses
- eGadget GUI
- Agent - resides on an independent platform and communicates with GAS-OS via sockets.
- The communication between the GAS-OS of two eGts is currently

eGts Interfaces

- *Sensor - FPGA board.* To implement these interfaces a generic signal conditioning board was designed. This board allows either digital or analogue voltages to be produced from the all the passive sensors that we have used. The circuit includes variable potential dividers, Schmitt trigger drivers and amplification to provide the LVTTL digital signal levels or analogue inputs in the range 0 to 3.3V.
- *iPAQ/laptop - WLAN.* PCMCIA expansion packs were used to connect the wireless cards to the iPAQs; these cards could be directly plugged into the laptops.

FPGA - iPAQ/laptop.

- A serial RS232 interface was created for the two-way communication between the FPGA and iPAQ/laptop. A UART function was programmed onto the FPGA to transfer parallel data into serial format with the required start-stop bits and data rates.
- A piggyback board was created to drive the RS232 serial cable from the LVTTL output of the FPGA and buffer the information being sent in the opposite direction. In the iPAQ serial port, Java classes and drivers from serialio.com were used to interface the serial data to the GAS-OS.

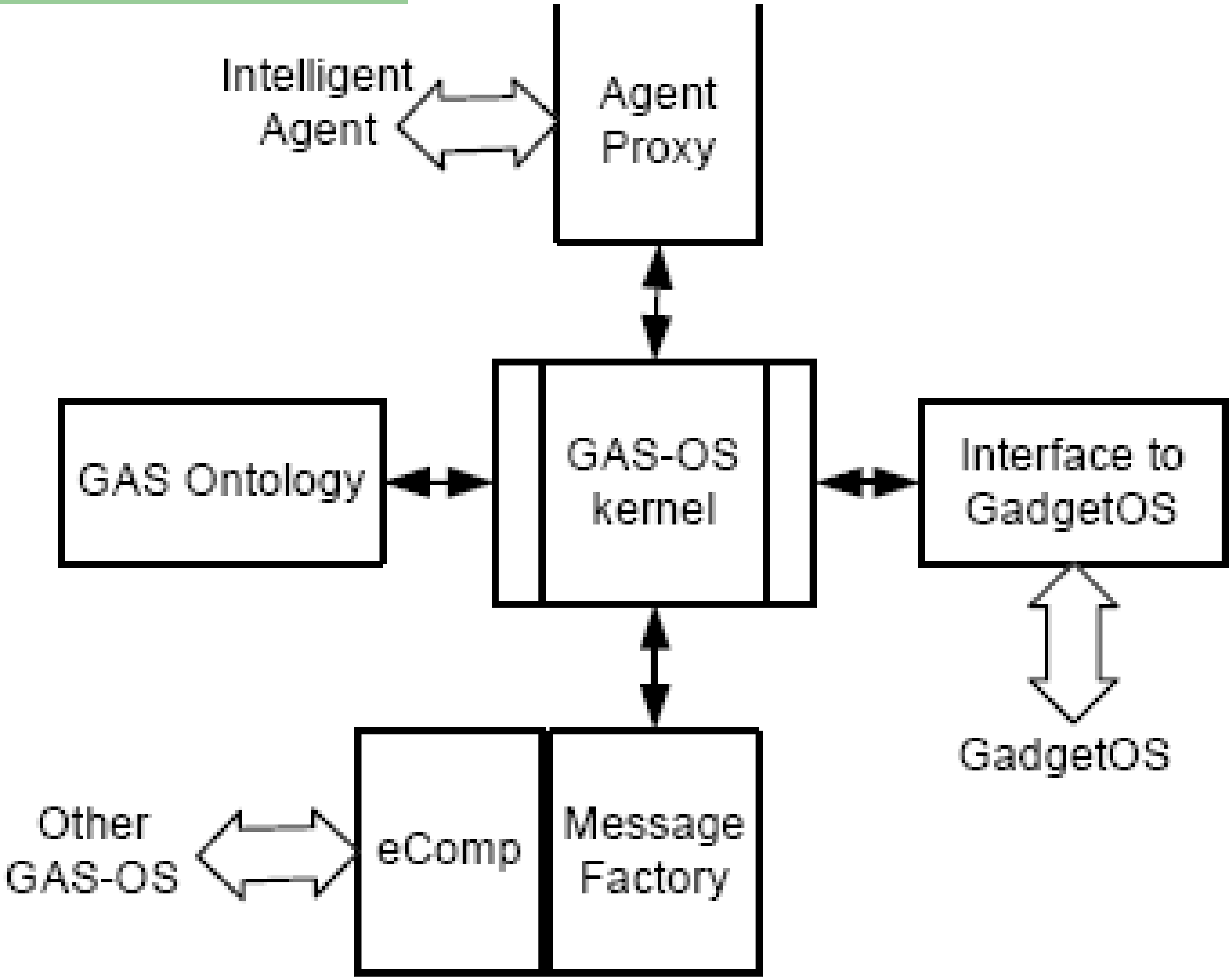
eGt - eGt..

- Cisco 350 series WLAN PC Cards allow the various eGts to communicate with each other. These cards work on the IEEE802.11b standard, which allows data transfer rates of up to 11Mbps. AD-HOC network mode was used to allow the eGts to communicate with each other without the use of an access point.

GAS Operating System

- GAS-OS is the middleware that runs on every eGt and implements GAS concepts.
- GAS-OS manages resources shared by eGts, and provides the underlying mechanisms that enable communication (and interaction) among eGts.
- Thus, it can be considered as a mini-operating system.
- **eGadgets** can be considered to be to GAS OS what **processes** are to operating systems.

GAS-OS ARCHITECTURE



Software/Hardware Modules

- The Gadget Management Software (GadgetOS) is responsible for providing access to the eGt resources (e.g. the RF unit, any sensors or actuators etc);
- The Collaboration Logic provides service discovery services;
- The Computation Logic implements the intrinsic eGt functions.

GAS Software Modules

- GAS-OS provides plug and synapse management services;
- Agent implements intelligent mechanisms.
- Requires two kinds of interface definitions: GadgetOS to GAS-OS and GAS-OS to Intelligent Agent/Computation Logic.

GAS-OS Services

- ***Plugs discovery and advertising**:* The GAS-OS of a specific eGt is responsible for the discovery of all other Plugs (and consequently eGts) within range.
- For this service, GAS-OS utilizes an '**extroverted Computing Platform**' (**eComP**) which implements the networking side of a P2P architecture for GAS. It multicasts a *hello message* and all eGts within range respond to it by sending an XML-based advertisement, which contains all the data that one eGt can know about the other, such as the list of SPlugs, the current IP address it has and the port it listens to, etc. Also, the GAS-OS gives the TPlug the ability to give to any connecting Plug the list of SPlugs that an eGt has.
- Given the fact that every eGt has a TPlug, the GASOS guarantees the accessibility of all the eGt Plugs.

eComp

- Handles the networking communication between the eGts.
- Peer-to-peer software module that enables the dynamic discovery and utilization of remote resources.
- eComP is implemented for the “J2ME + CDC profile” platform
- Based on message exchanging between remote peers of a network. No infrastructure is assumed to be present, except for TCP/IP networking.
- Discovery of remote peers is performed using multicast socket connections, thus it does not require some sort of central infrastructure.
- eComP does not expect the connected peers to be statically bound to a specific IP address. Communication is based on the unique eComP IDs that peers have, which remain the same even when peers may switch networks and utilize new IPs.

Synapse establishment – disestablishment

- GAS-OS enables the user to form or destroy Synapses between Plugs.
- It ensures that Plugs get connected only when they are available and “compatible” (an ontology is used to define the degree of compatibility).
- Then, it takes care of the handshaking between the two connecting Plugs until the Synapse is established.
- GAS-OS provides the means for successful connection or disconnection, ensuring that these procedures are executed as atomic ones that either succeed or fail before releasing the Plugs.
- Moreover, it ensures that the Plugs will not stay locked for infinite amount of time, in the case where a Synapse establishment fails.
- The Plugs are fully functional and do not stay locked during this procedure. This is very important because network delay can be long even for successful Synapse establishments.
- Also, it is the GAS-OS responsibility to ensure that when an eGadget is shutting down, all connected Plugs are notified.
- Finally, the GAS-OS ensures that on startup an eGt will attempt to reestablish the Synapses of the eGWs it participated in when it was shutdown.

message factory:

- Since GAS-OS utilizes message-based communication, all data exchange is performed using formatted messages.
- Responsible for encoding and decoding GAS-OS messages.
- In order to make the communication procedure easier and to maximize code reuse, this module was developed so that the standard part of XML formatting and decoding that is necessary for the communication is not coded into every single GAS-OS class.
- The message factory module is implemented for the Personal Java platform.

GadgetOS interface:

- Handles communication between the GAS-OS and the GadgetOS of the eGadget.
- Offers a standard interface through which the GAS-OS can “talk” to the eGadget specific code that the manufacturer implements.
- Thus, updating of the eGadget specific code is easy. (The critical issue here is that the manufacturer ensures that this code is compatible with Personal Java.)

… Services

- ***Agent Proxy:*** the module that is attached to and collaborates with the Intelligent Agent. It is implemented for the Personal Java platform.
- ***GAS-OS kernel:*** the central module of the GAS operating system.
- Responsible for initialization and management of Plugs and implements the services offered by the GAS-OS.
- Implemented for the Personal Java platform.