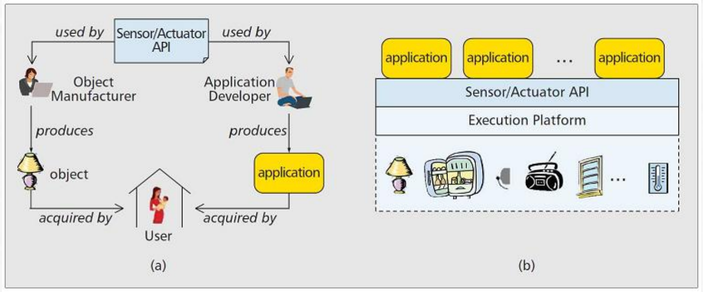
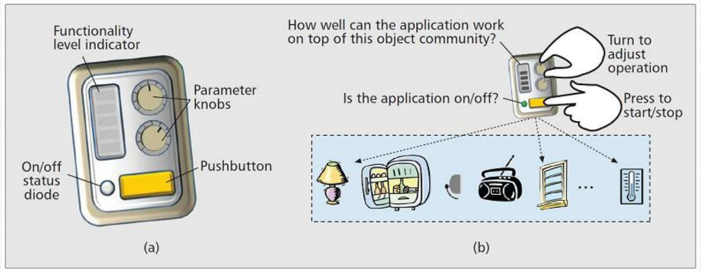
GRASPABLE AND RESOURCE-FLEXIBLE APPLICATIONS FOR PERVASIVE COMPUTING AT HOME

reifying smart home applications …

to make them more like a flashlight




Functionality level indicator
Parameter knobs
Pushbutton
On/off status diode
(a)
How well can the application work on top of this object community?
Turn to adjust operation
Is the application on/off?
Press to start/stop
(b)

Prepared by:

Jaroslaw Domaszewicz
Institute of Telecommunications
Warsaw University of Technology
Warsaw, Poland

Spyros Lalis
IRETETH/CERTH & University of Thessaly
Volos, Greece

Based on initial sections of the paper:

Domaszewicz, J.; Lalis, S.; Paczesny, T.; Pruszkowski, A.; Ala-Louko, M.
Graspable and resource-flexible applications for pervasive computing at home
IEEE Communications Magazine, IEEE , vol.51, no.6, pp.160-169, June 2013

Part I. Ecosystem for resource-flexible pervasive computing at home

Sensor/Actuator API

Our vision of pervasive computing for the domestic environment assumes an ecosystem formed by an API and three basic stakeholders (Fig. 1a).

The **sensor/actuator API** captures relevant resources in the form of primitives.

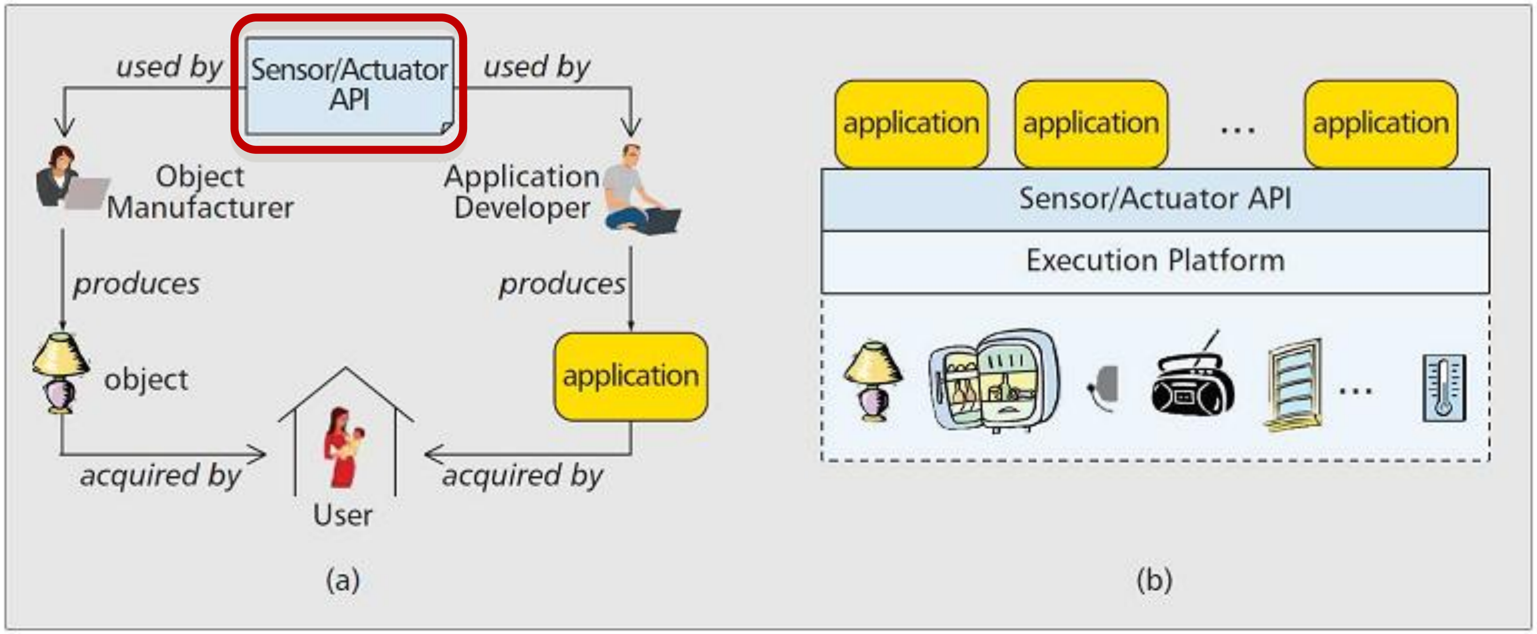


Figure 1. a) Key stakeholders involved in the introduction of smart objects and applications in the home; b) at home, objects with different sensors and actuators jointly form a platform for the execution of applications.

Objects and applications are developed independently

Object manufacturers produce smart objects, which, in addition to their regular functionality, expose their sensors and actuators through appropriate API primitives.

Application developers pick the primitives needed for sensing and actuation in their smart home applications.

The processes of object and application development are totally **decoupled.**

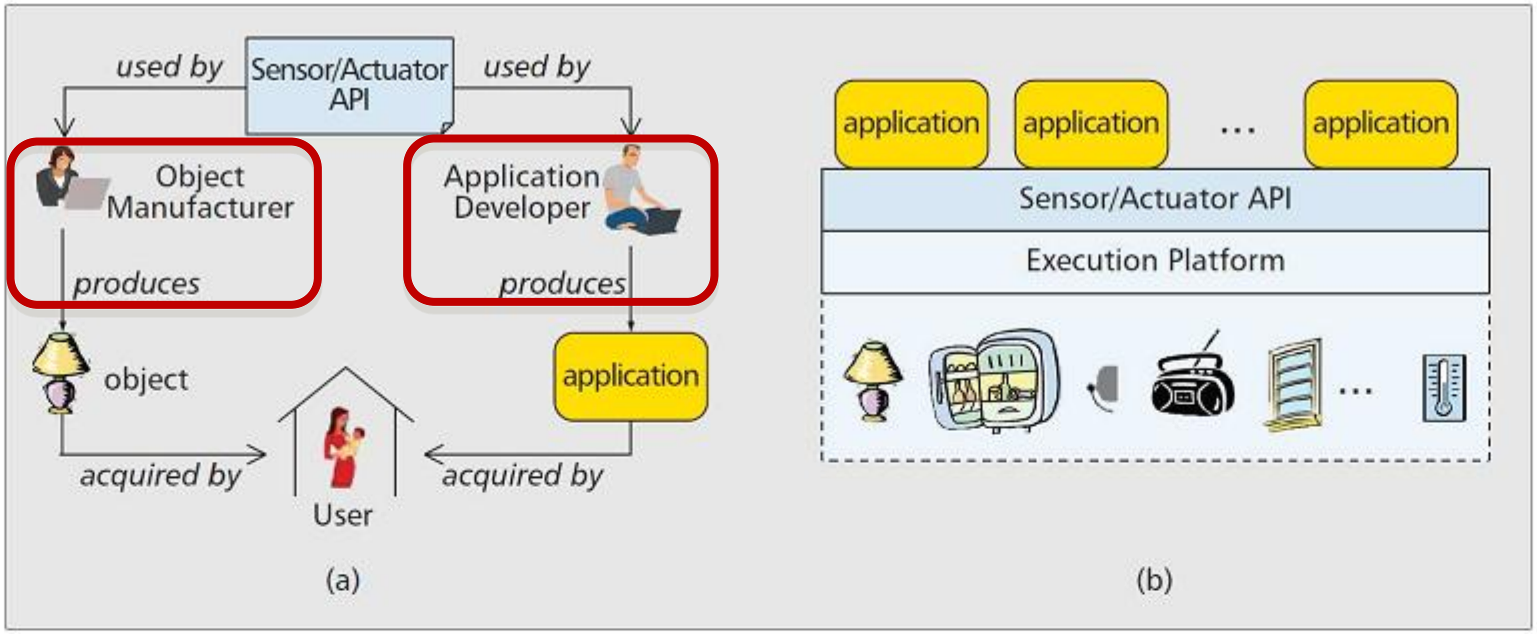


Figure 1. a) Key stakeholders involved in the introduction of smart objects and applications in the home; b) at home, objects with different sensors and actuators jointly form a platform for the execution of applications.

The smart home platform is formed ad-hoc

The user purchases objects for their regular functionality. At home, objects jointly form an application execution platform (Fig. 1b).

The platform emerges as a **side-effect** of populating the home with objects that are useful for its inhabitants **as such**, without consideration for the applications.

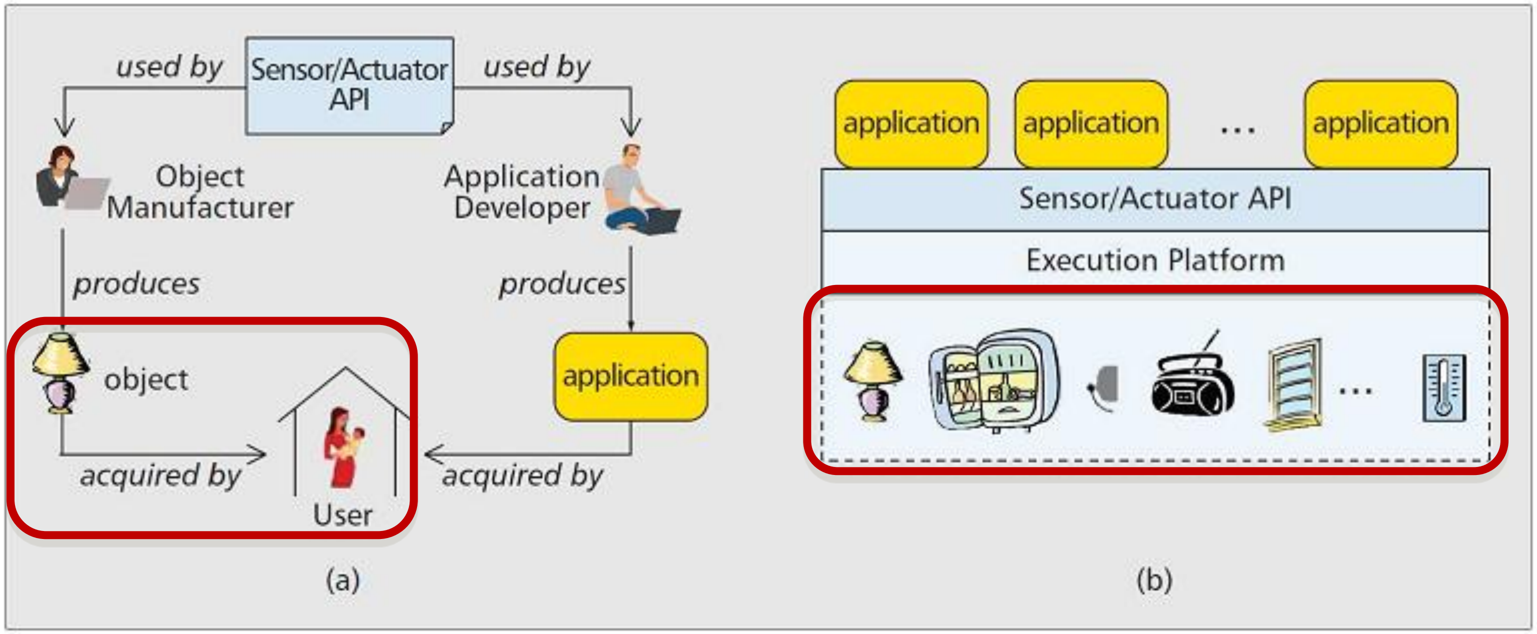


Figure 1. a) Key stakeholders involved in the introduction of smart objects and applications in the home; b) at home, objects with different sensors and actuators jointly form a platform for the execution of applications.

Applications are acquired without thinking about available objects

In a similar vein, the user acquires applications without knowing how well the object collection in his home can actually support them.

Of course, these applications cannot be life critical.

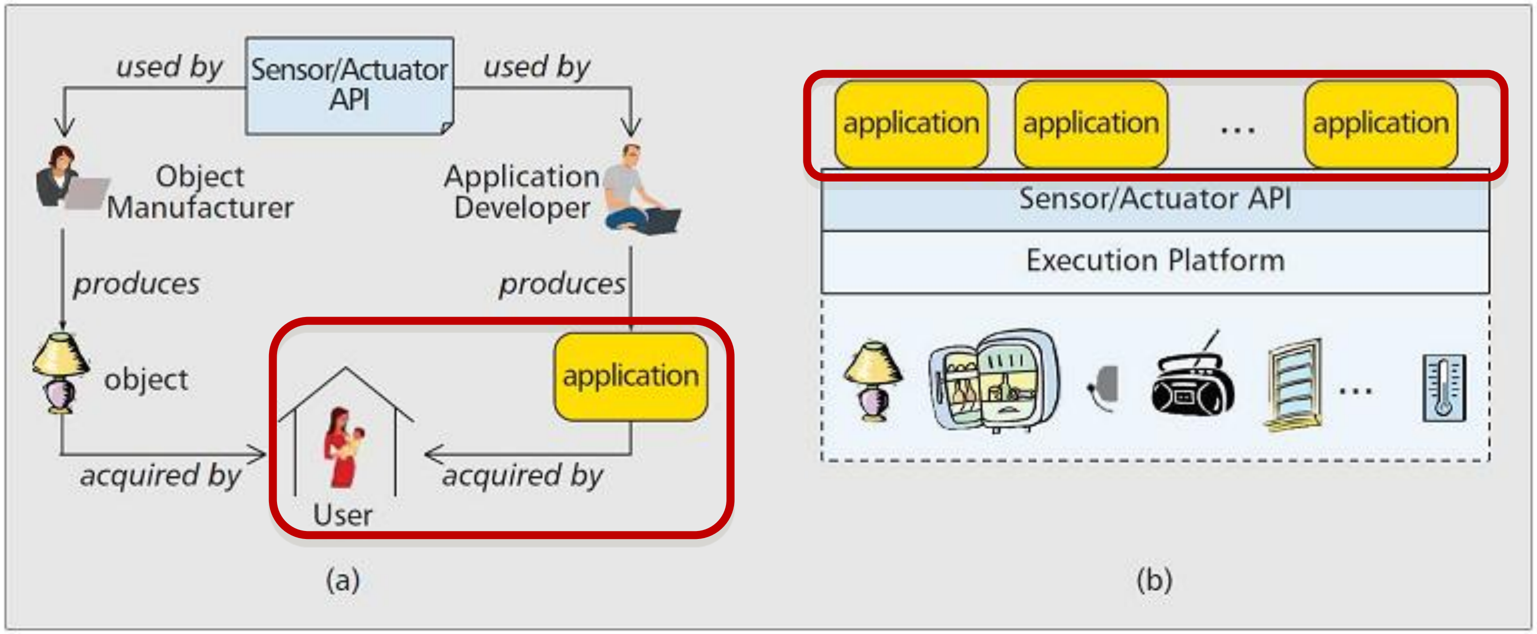


Figure 1. a) Key stakeholders involved in the introduction of smart objects and applications in the home; b) at home, objects with different sensors and actuators jointly form a platform for the execution of applications.

Result: sensor/actuator resources are unknown at the development time

The “unplanned” acquisition of objects implies that it is unlikely for any two homes to feature the same object collection.

It follows that the objects (sensors and actuators) of the target platform are **unknown when the application is developed.**

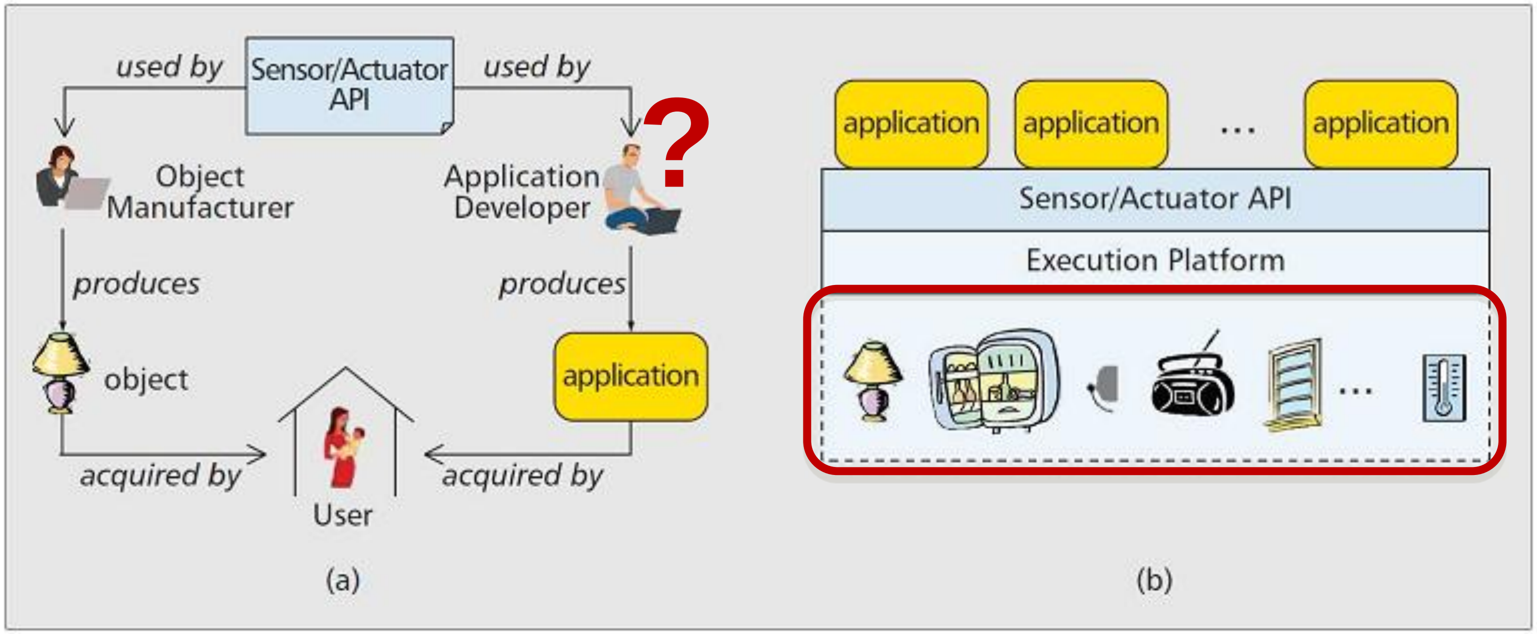


Figure 1. a) Key stakeholders involved in the introduction of smart objects and applications in the home; b) at home, objects with different sensors and actuators jointly form a platform for the execution of applications.

Solution: resource-flexible applications

If the application is to be of value across a wide range of homes, it must be **resource-flexible**, i.e., it should function (instead of quitting) even if it is lacking some sensors and actuators.

Resource flexibility comes with uncertainty about how well the application will actually work;
it may be able to deliver its functionality only partially.

If so, the application should inform the user about **the functionality it can deliver in the home where it is deployed**.

Part II. The application pill: a graspable application for resource-flexible computing

Reifying applications to make them more like a flashlight

We argue that the best way for the application to “disappear” is (somewhat paradoxically) **to reify it as a regular object** that blends into the domestic environment.

As a yardstick, think how **easy it is for anyone to operate a flashlight**: it only has an on/off switch, and it is trivial to check if it works, just by glancing at it. We wish this to hold for smart home applications as well.

To this end we propose the concept of the **graspable application**: a small physical artifact that embodies a single application and features an absolutely minimal interface (without any general-purpose UI elements) for controlling and monitoring its operation.

The point is for the user to conceptually identify the application itself with a physical object, which can be grasped and manipulated as easily as the flashlight.

Envisioning the graspable application (1/3)

We envision the graspable application as a **small object, roughly the size of a matchbox**, as shown in Fig. 2.

The object has a pushbutton to start/stop the application and a diode to show whether the application is running.

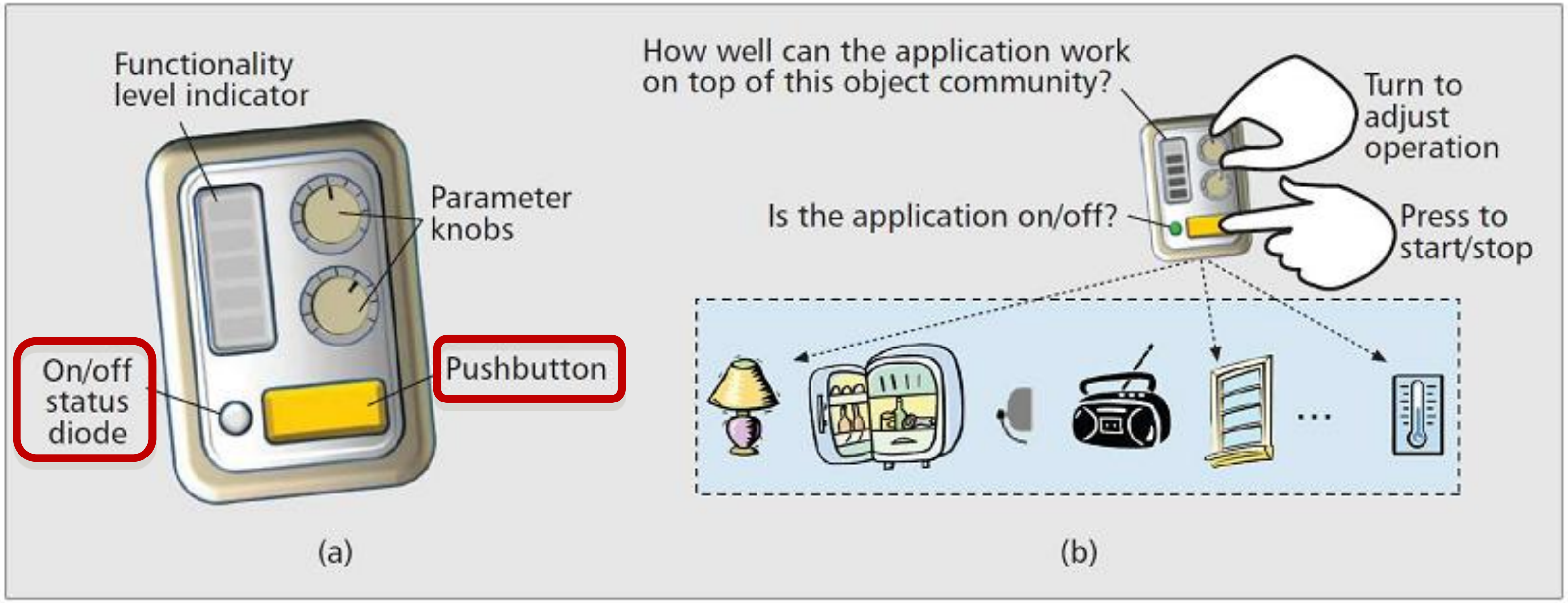

Functionality level indicator
Parameter knobs
On/off status diode
Pushbutton
(a)
How well can the application work on top of this object community?
Turn to adjust operation
Is the application on/off?
Press to start/stop
...
(b)

Figure 2. The "canonical" application pill design (objects are not shown to scale): a) the application pill object, featuring a minimal interface to control and monitor the resource-flexible application; b) the application is started/stopped by pressing/depressing the button; the diode shows the on/off status of the application; the knobs are used to set application parameters; the linear indicator shows the functionality level of the application.

Envisioning the graspable application (2/3)

We capture the **functionality level** of the application as a fraction of the functionality it would be able to provide if it had at its disposal all the sensors and actuators it requests from the underlying object community.

The functionality level can be intuitively displayed via a simple gauge, like the ones used to show the signal strength of wireless devices.

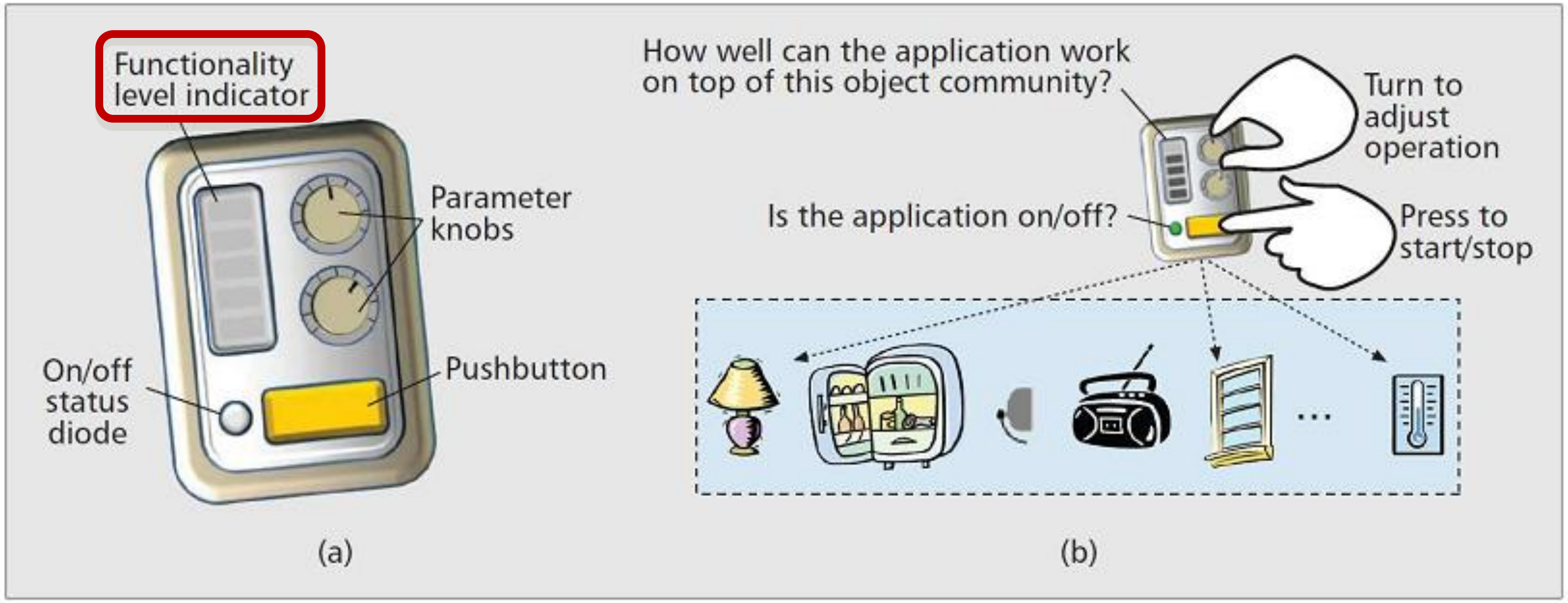

Functionality level indicator
Parameter knobs
On/off status diode
Pushbutton
(a)
How well can the application work on top of this object community?
Turn to adjust operation
Is the application on/off?
Press to start/stop
...
(b)

Figure 2. The "canonical" application pill design (objects are not shown to scale): a) the application pill object, featuring a minimal interface to control and monitor the resource-flexible application; b) the application is started/stopped by pressing/depressing the button; the diode shows the on/off status of the application; the knobs are used to set application parameters; the linear indicator shows the functionality level of the application.

Envisioning the graspable application (3/3)

The application object may feature one or more knobs, each for setting a single parameter (e.g., the setpoint for a temperature control application).

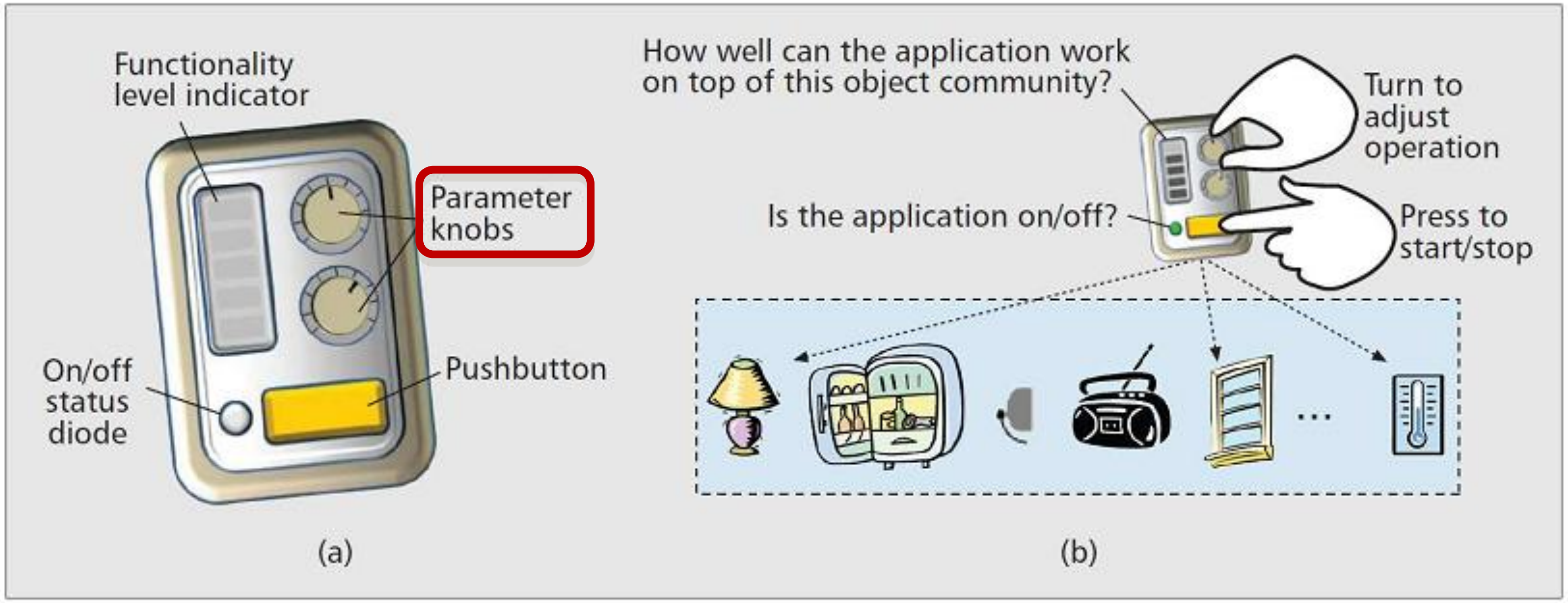

Functionality level indicator
Parameter knobs
On/off status diode
Pushbutton
(a)
How well can the application work on top of this object community?
Turn to adjust operation
Is the application on/off?
Press to start/stop
(b)

Figure 2. The "canonical" application pill design (objects are not shown to scale): a) the application pill object, featuring a minimal interface to control and monitor the resource-flexible application; b) the application is started/stopped by pressing/depressing the button; the diode shows the on/off status of the application; the knobs are used to set application parameters; the linear indicator shows the functionality level of the application.

The application pill (1/2)

To stress that such a graspable application object should be truly minimal (in terms of both size and interface), and to hint that it carries the application's code, we call it the **application pill**.

A pill could be accompanied by a one-page manual describing what the application will do for the user at different values of the functionality level.

From the user's perspective, the application pill is the application. It can be casually placed anywhere in the home (Fig. 3).

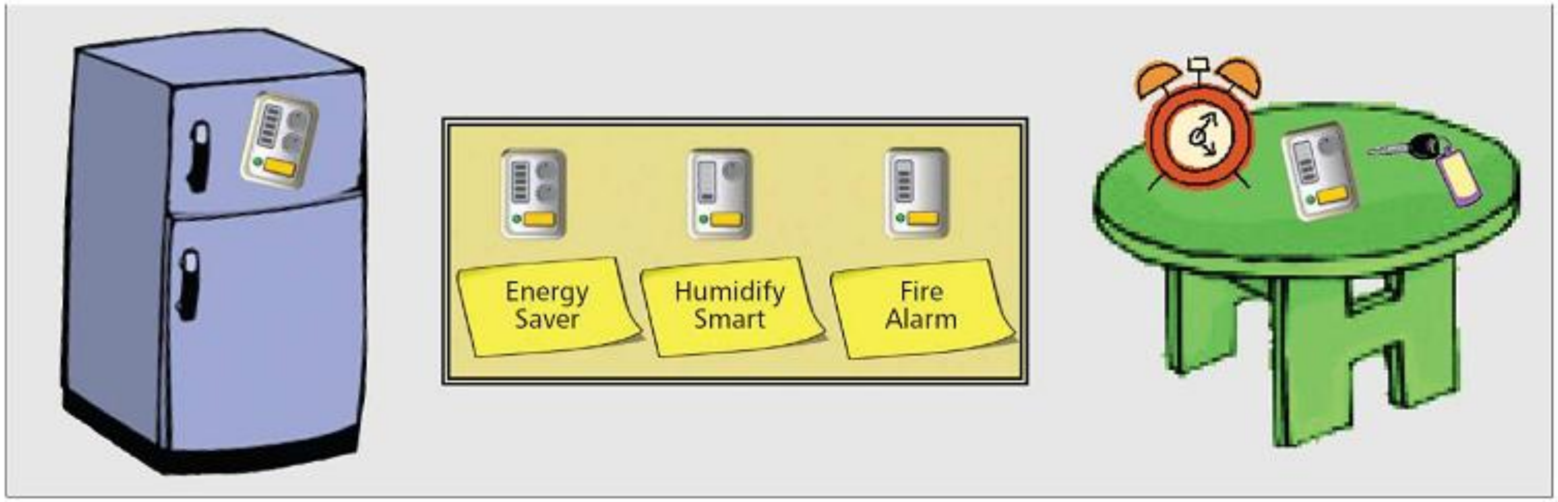


Figure 3. Application pills can be put at various convenient places in the home (objects are not shown to scale).

The application pill (2/2)

Of course, one can imagine several variations of our “canonical” application pill design (Fig. 4). However, it is important to note that the application pill should not be transformed into yet another attention-hungry computer-like device.

If the application has to support complex input/output functions, these should be delegated to devices with a powerful interface, likely to be found in the home (say, a TV or smartphone).

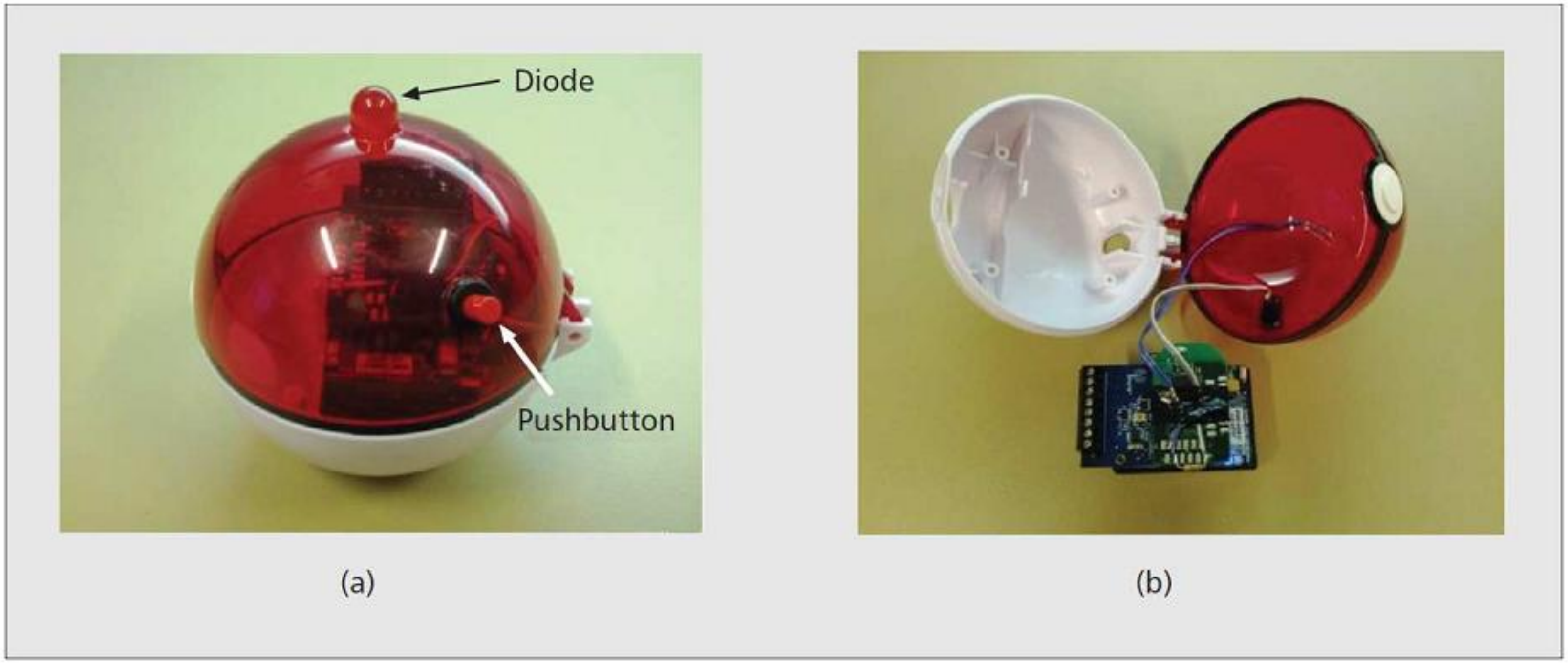
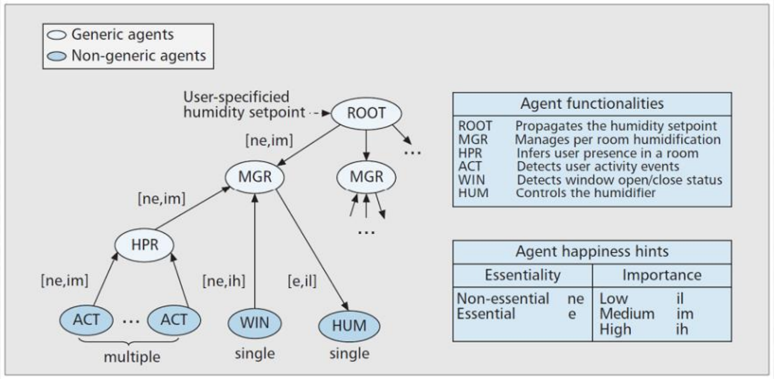


Figure 4. The POBICOS application pill: a) the user view: a pushbutton for starting/stopping the application, and a diode that shows the on/off status as well as the functionality level of the application; b) the internal view: the Imote2 (with the application binary and the POBICOS middleware) wired to the pill's button and diode.

This work was funded in part by the 7th Framework Program of the European Community, project POBICOS, contract no. FP7-ICT-223984. Thanks to Manos Koutsoubelias, Markus Taumberger, Tomasz Tajmajer, and Vladimir Palacka.

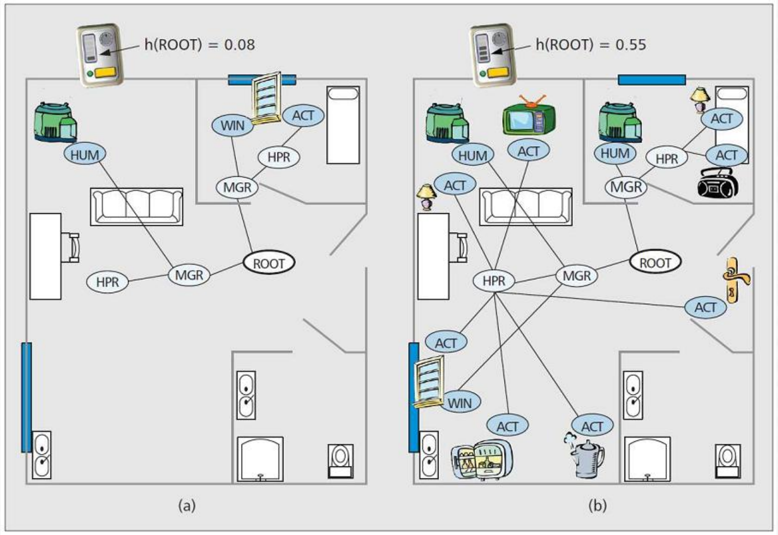
For the full story, go to

Domaszewicz, J.; Lalis, S.; Paczesny, T.; Pruszkowski, A.; Ala-Louko, M.
Graspable and resource-flexible applications for pervasive computing at home
IEEE Communications Magazine, vol.51, no.6, pp.160-169, June 2013,
doi:10.1109/MCOM.2013.6525610



Agent functionalities	
ROOT	Propagates the humidity setpoint
MGR	Manages per room humidification
HPR	Infers user presence in a room
ACT	Detects user activity events
WIN	Detects window open/close status
HUM	Controls the humidifier

Agent happiness hints			
Essentiality		Importance	
Non-essential	ne	Low	il
Essential	e	Medium	im
		High	ih



Some further items in the paper:

1. a tree-based programming model with resource-driven, partial tree instantiations,
2. an API for programmer-supported hints that allow runtime calculations of the functionality level,
3. HumidifySmart – a case study of a resource-flexible application with two deployment examples.