

### Smart-Its Features

We will illustrate certain Smart-Its enables features with three different kinds of food; wine, cheese and oysters in our presentation of Smart-its at the Jamboree. A number of real food pieces (or perhaps plastic models) will all have Smart-its attached to them. The supporting software architecture will however not differ much between the different food types.

In principle, we would only have to specify a Java API that we would like to have in order to implement the higher level application, including displays for visitors, short explanatory video pieces and animations, etc. This API includes ways of querying certain sensor values as well as more complex percepts. However, to make things more clear, we explain here how we imagine that the percepts might be implemented.

### Parts of the system

In this scenario, we will use a number (approximately 5-10) mobile, wireless Smart-Its-augmented food objects. We will also have 2 gateways, i.e. Smart-Its RF boards connected to a host PC via a serial cable. The purpose of these gateways is to relay data packages to and from the PC to the wireless nodes in the ad-hoc network. One of these gateways will be out in an open space, with a range of about 5-10 meters. Another gateway will be installed inside a refrigerator. When the fridge door is closed, only the gateway inside the fridge can communicate with the objects inside the fridge. When the fridge door is open, both gateways should be able to communicate with the objects inside the fridge, and vice versa for the objects outside the fridge. Hopefully this can be accomplished by means of using the fridge as a Faraday's cage with a metal shield effectively hindering the RF traffic to penetrate it when the door is closed. If we don't get this to work for real with a standard fridge, we can still "fake" this behaviour to get the same higher-level behaviour using a door switch.

Minimum number of Smart-Its objects we need:

- 2 Oysters
- 1 Cheese/Plate - the cheese and its plate will have the same smart-it
- 1 Cheese slice
- 2 Wine bottles
- 1 Tray maybe needed to get additional data for the grouping percept.
- 1 Fridge gateway
- 1 Room gateway

Total: 7 + 2 gateways. Possibly we will have a few more wine bottles etc if everything works fine.

### Communication

The communication between the Java API and the wireless Smart-its should be a two-way communication where the Java program could ask for specific information packages, or subscribe to certain packages (with sensor data) at a set interval. One reason for this is to minimize wireless network traffic.

Typically, all sensor value communication is initiated from the PC, either by sending an explicit query to one or more Smart-Its (addressing a particular Smart-It ID number, a class of Smart-Its such as all wine bottles, or a generic broadcast to all Smart-Its within range). Here, the addressed Smart-Its would respond as quickly as possible (perhaps using some algorithm to lower RF packet collisions due to several Smart-Its responding at the same time), return the requested sensor value back to the PC via a gateway Smart-It.

Another way to get sensor values would be to subscribe for sensor readings from one or more sensor types at a regular interval (e.g. once every second). This type of communication would still be initiated in the same way as above. Of course there should also be away to change the interval or to disable a stream (e.g. set the interval value to 0).

A few special events, like shaking or abrupt movement should be sent to the Java API immediately. To implement this there must be code running of each Smart-It that continuously checks is one or more of these special conditions are met, and if so an alert is sent to the Java program. The Java API should include functions for changing the threshold values that triggers these alarm events.

All data communication should pass through two gateway smart-its, connected to the pc using two serial ports. By including a refrigerator with one of the gateways we hope to make an area sealed off from the rest of the room while closed. This will create a difference between the smart-its, in the room and in the fridge and will speed up calculations. Another communication aspect is that the gateways can handle the minimum number of smart-its that will be used without losing too many packages.

### percepts that we need through the API

- **Basic sensor readings** from all sensors as raw data (in the form of bytes and byte arrays).
- **Grouping percept** - this percept will be used to group different smart-its to each other when moved on a plate. We suggest that accelerometers will be used for this - similar to how the "Smart-Its friends" work. The smart-its could be fixed in a specific position on the tray or plate to get the x/y axes in the same direction. If this is possible, you could also use sensor fusion between accelerometers and light, and possibly also sound. The sensor fusion, in that case, and the actual grouping would be done in the java program. The general idea is that a chef or some other human groups a number of objects together, in this case as a finished dish to be served, by moving the tray from one place (a table where the food is prepared on the plate) to another (a shelf where finished orders are placed). When moving the entire plate, with its onboard, smart-its-tagged objects, all these objects will have a similar trajectory through the context space (an abstract mathematical space made up of all available sensor readings and percepts), and therefore it should be able to distinguish these objects from the other smart-its in the room. All the moved objects will for a second or two have almost identical readings with respect to acceleration (X and Y axis) as well as light readings and perhaps also sound level readings. We will of course design our demonstration space so that there is a substantial difference in light level between the table and the shelf.
- **Ready to be served** - this percept is a combination of grouping and reaching the right temperature level for all objects in the group. This is a very high level percept that depends much on several particular pieces of food as well as other more abstract things like menus and recipes. So, even though this can be seen as a percept, it should probably be implemented by the application code written by PLAY rather than being included in the Java API written by PCCV.
- **Decay** - both the wine, oysters and the cheese will have a lifecycle with decay. The decay will be a mathematical function of several variables such as time, temperature, movement, and will of course differ between different food types. This will be a higher level percept based on shaking, storage angle (for wine) and temperature. The details of this function will be decided later (partly empirically by testing different threshold values for acceleration reading etc). These decay functions needs to be part of the Java Percepts API and its underlying parts since some of the higher level percepts will depend on the decay values as well as direct, real-time sensor readings.

### Cheese

- **Cheese running out** - The cheese/plate should feel the pressure change when a cheese slice is taken of. The pressure sensor should be attached either on the plate, hidden under the cheese, or perhaps we will drill a hole in the plate, had the smart-it inside the cheese and have the pressure sensor on the downside of the plate (so that it measures the pressure between the plate and the table). In effect, the pressure sensor will work as a primitive scale, measuring the weight of the plate and the cheese on it.

### Wine bottle

- **Shaking** - The wine bottle senses when being shaken which affects the decay process. This is a derivative of acceleration sensor data.
- **Angle** - The wine bottle must tell which angle it presently has (with 5-10 degrees precision or something like that), lying down or standing up which affects the decay process. This is a derivative of acceleration sensor data.

### Fridge

- **Fridge, open or closed** - This percept can simplify later calculation and will also be used to trigger video scenarios. Hopefully it will be possible to sense this percept when using the fridge as a Faraday's cage: e.g. when the door is open the two gateways can communicate wirelessly (by sending a small ping package or something), and when the door is closed they will be unable to communicate because the gateway inside the fridge will be physically shielded from the rest of the room. If this doesn't work well, we can still get the same behaviour by using a small door switch embedded into the fridge. From a higher level view, both ways of sensing will be identical.

Object	Percept	Sensors	Output
All smart-its	Basic sensor readings	All	Raw data
Cheese slice	Grouping	Accelerometers (light) (sound)	Grouped objects
	Ready to be served	Grouping Temperature	Yes/No
	Decay	Temperature	Degrees celsius
Cheese	Cheese running out	Pressure	
Wine bottle	Grouping	Accelerometers (light) (sound)	Grouped objects
	Ready to be served	Grouping Temperature	Yes/No
	Decay	Shaking (accelerometers)	Yes/No
		Angle (accelerometers) Temperature	Lying/Standing Degrees celsius
	Shaking	Accelerometers	
Angle	Accelerometers		
Oysters	Grouping	Accelerometers (light) (sound)	Grouped objects
	Ready to be served	Grouping Temperature	Yes/No
	Decay	Temperature	Degrees celsius
Fridge	Fridge door status	Light? Connection to outside gateway?	Open, closed