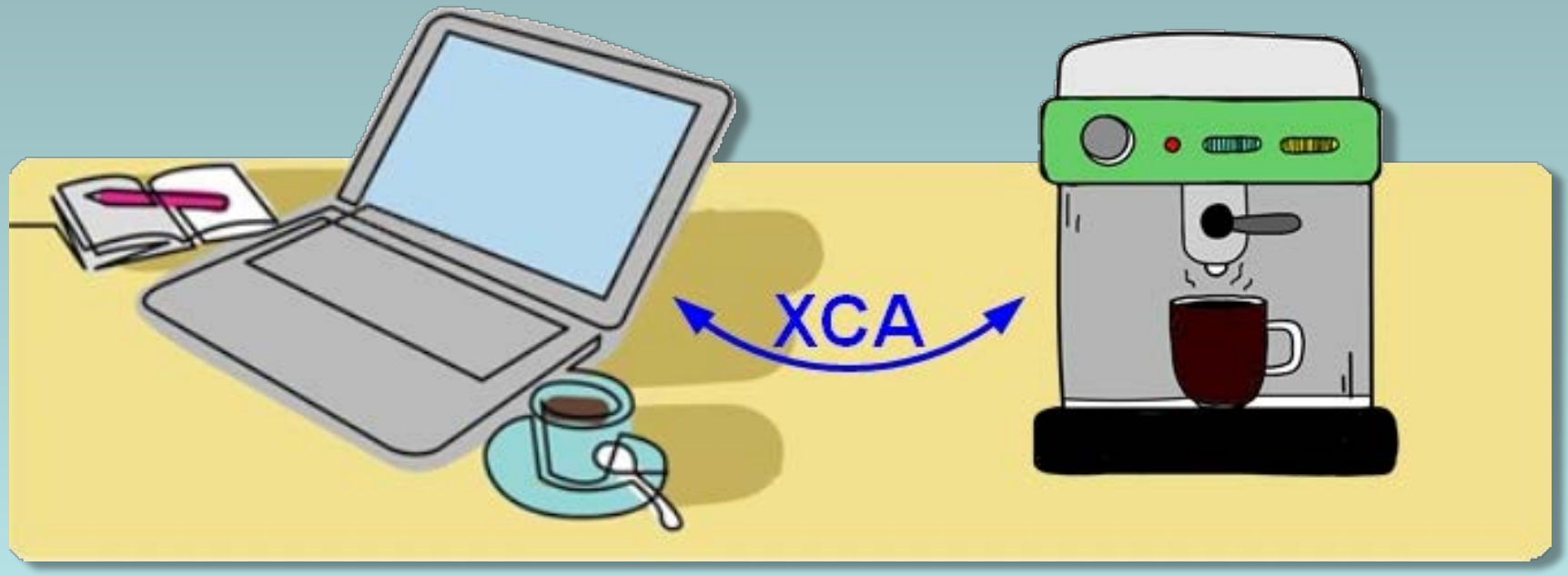
XCA
eXample Control Architecture

A tiny tutorial example that introduces key concepts of OCA,
the Open Control Architecture

rev 06 / 2020.05.14

XCA - Example Control Architecture

- XCA ("eXample Control Architecture") is a tutorial concept to introduce key concepts of OCA, the **Open Control Architecture**.
- XCA is a highly simplified version of OCA.
- XCA is not meant to be implemented -!
- This following pages define XCA and show how it could be used for computer control of a simple single-cup espresso machine.



XCA - Example Control Architecture

XCA

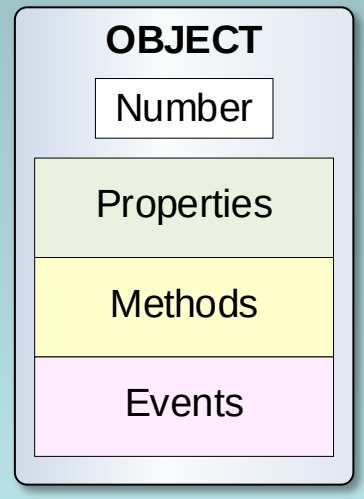
XCA - Example Control Architecture

Object Orientation

- XCA is an ***object-oriented*** control scheme. This means:
  - A device is controlled by a set of control (/monitoring) ***objects***.
  - **An object is a network interface**.
  - Each object defines a mini-API for network control of a device function.
  - The device's network control API is simply the sum of the mini-APIs of all its objects.
  - XCA objects are NOT software objects inside devices. They are network interface definitions only! Device internal software structure is outside XCA's scope.

- ***What's in an XCA object?***

1. Every object is identified by an **object number** that's unique within the device.
2. An object has three kinds of components:
   a. One or more **properties.** A property is a device operating parameter.
   b. One or more **methods.** A method is a procedure that a controller may call to change a property value or do other things.
   c. Zero or more **events.** An event is a tiny intelligence whose job is to notify a controller of something that has happened: a property value has changed, a state has changed, etcetera.



XCA - Example Control Architecture

Classes

- Objects are constructed from standard templates called **control classes**.
- The set of all XCA control classes is called the **XCA control model.**
- Each class's definition is based on the definition of another (*parent*) class, and is said to *inherit* the elements of that class. One class has no parent - the *root class*, the common ancestor of all other classes.
- XCA is a simple example without many classes. They are:

xcaRoot	the root class
xcaSwitch	controls an n-position switch or n-position option selector
xcaFractionalValue	controls a numeric value between 0 and 1
xcaTemperature	controls a temperature setting
xcaStateSensor	monitors an internal state
xcaFractionSensor	monitors a numeric value between 0 and 1

- The next slide details the properties, methods, and events of these classes.

XCA - Example Control Architecture

Class Details

Class	Properties *(c datatypes in italics)*	Methods	Events
xcaRoot	*string* Name readonly name of function in device - All classes will inherit this property	GetName(Name)	
xcaSwitch	*uint8* Position range = 0...255	GetPosition(Position) SetPosition(Position)	
xcaTemperature	*float32* Value any value	GetValue(Value) SetValue(Value)	
xcaStateSensor	*uint16* Value range = 0...65536	GetReading(Reading)	PropertyChanged notification sent to controller when **Reading** changes value
xcaFloat32Sensor	*float32* Value	GetValue(Value)	PropertyChanged notification sent to controller when **Reading** changes value

XCA - Example Control Architecture

Protocols

XCA - Example Control Architecture

XCA Protocols: *What actually goes along the wire?*

Recall

- Each control object defines a mini-API.
- A device's network control API is the sum of the mini-APIs of all its objects.

Therefore

- **What's needed is a protocol that can handle XCA's mini-APIs.**

Here's what goes along the wire

1. Controller calls to XCA methods in devices
2. XCA device responses to controller method calls
3. Event messages from XCA devices to controllers

Commands
Responses
Notifications

Any protocol that can reliably transport commands, responses, and notifications should work with XCA.

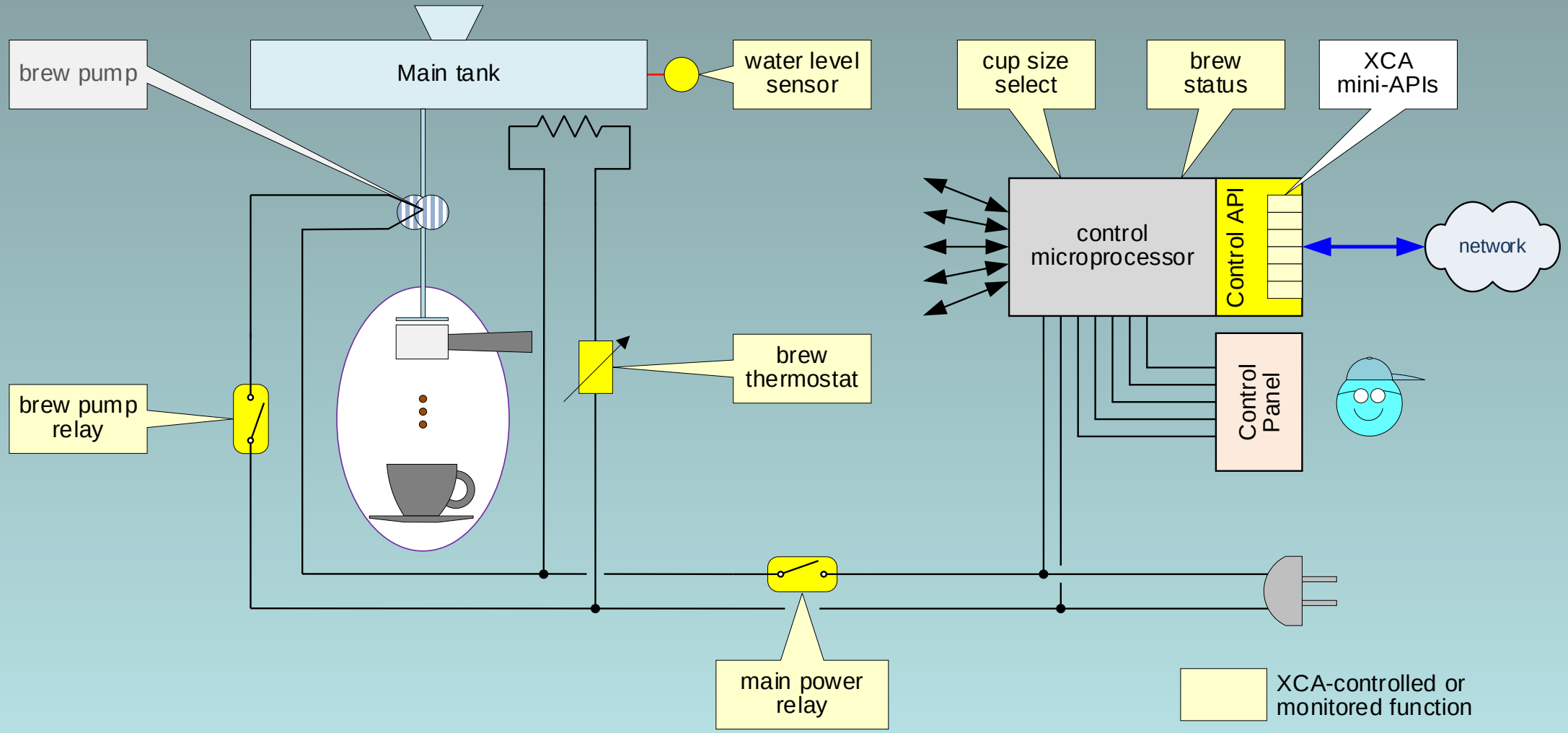
For now, let's just assume we have an appropriate XCA protocol.

For the curious, **Appendix 1** shows how a protocol works with classes and objects.

Example:
Single-Cup Espresso Machine

XCA - Example Control Architecture

Espresso Machine



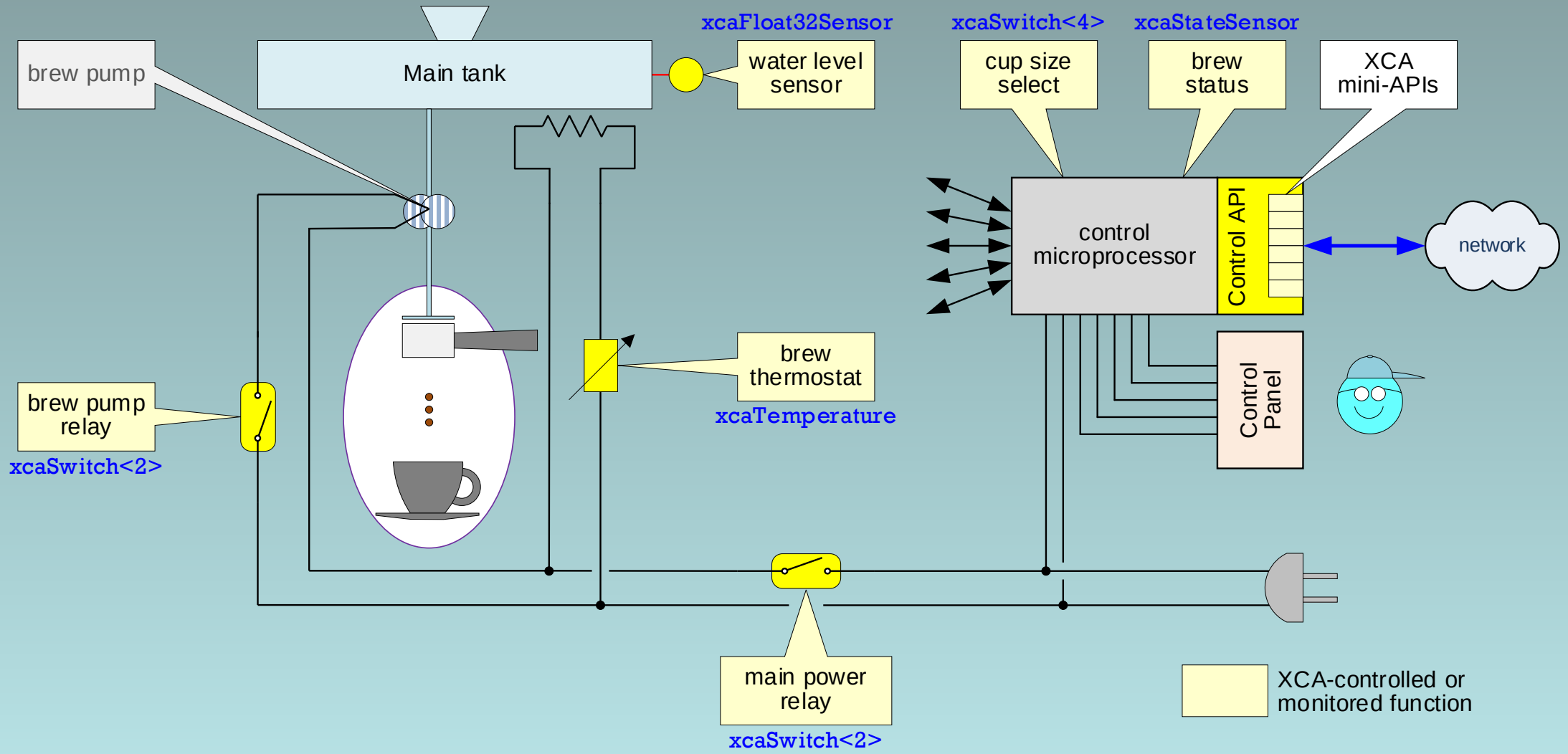
XCA - Example Control Architecture

Espresso Machine Objects

Obj #	Device Function	Control Class	Name	How it works
1	Device off(0) / on(1)	xcaSwitch (2 positions)	MainPower	Turns power on when set to 1
2	Tank water level	xcaFloat32Sensor	TankLevel	Reads out water level. Value ranges from 0 (empty) to 1 (full).
3	Brew status	xcaStateSensor	Status	0=power is off, 1=tank empty, 2=temp too low, 3=ready to brew, 4=brewing, 5=machine error
4	Brew pump off(0) / on(1)	xcaSwitch	Brew	Turns brew pump on when set to 1
5	Brew water thermostat	xcaTemperature	BrewTemp	Value determines setpoint of water level thermostat.
6	Cup size select (0, 1, 2, 3)	xcaSwitch (4 positions)	CupSize	Sets microcontroller software for desired cup size.

XCA - Example Control Architecture

Espresso Machine, showing classes of XCA objects assigned to each control function.



Further Reading

XCA - Example Control Architecture

Further reading

- Information on the *real* OCA and about AES70, the Audio Engineering Society media device control standard based on OCA, can be found online, in these places:
  - OCA Alliance general website, http://oca-alliance.com/
  - OCA Alliance technical website:
    - ***Downloads*** page, https://ocaalliance.github.io/downloads.html
    - ***Developer Resources*** page, https://ocaalliance.github.io/resources.html
  - Audio Engineering Society standards:
    - **AES70-1, AES70 Framework,**
    - **AES70-2, AES70 Class Structure,**
    - **AES70-3, AES70 Protocol for TCP/IP Networks.**

    Concise instructions for obtaining copies of these standards may be found at https://ocaalliance.github.io/resources.html

Appendix 1:

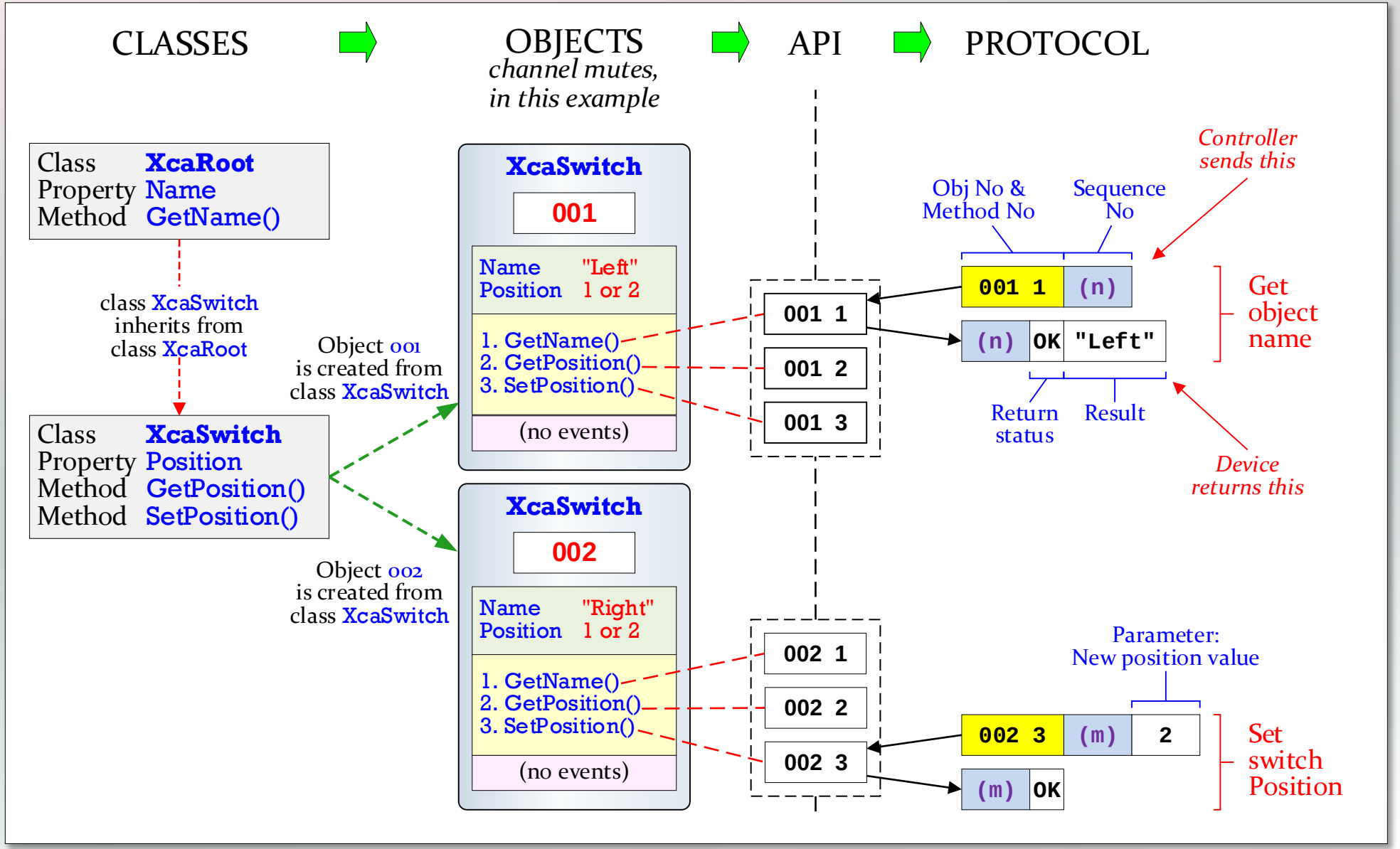
How a protocol works with classes and objects

Optional material for detail-oriented readers

XCA - Example Control Architecture

How a protocol works with classes and objects

- *Classes define objects.*
- *Objects define APIs.*
- *APIs inform the protocol.*



Appendix 2:

Selected operation sequences

Optional material for detail-oriented readers

XCA - Example Control Architecture

Operation Sequences - 1

- Startup

1. User starts espresso machine control app in computer connected to the espresso machine's network.

2. Control app finds espresso machine microprocessor on the network by magic.
   **** Well … not really by magic, but by a process we won't describe here.*

3. Once connected to the espresso machine's microprocessor, controller calls **Status.GetValue(…)** to discover machine status.

4. Depending on machine status, control app may
   - Power up the machine using **MainPower.SetValue(…)**
   - Ask the user to add water, and check the result using **TankLevel.GetValue(…)**.
   - Abort the operation, is machine status value is 5 (error).
   - If operation is not aborted, controller repeats steps 1 and 2 until machine status value is 3 (ready to brew).

5. If/when all is well, control app displays ready-to-brew message.

XCA - Example Control Architecture

Operation Sequences - 2

- Brewing

1. User selects cup size (or it defaults). Controller calls **CupSize.SetValue(...)** to inform the espresso machine microcontroller of the chosen cup size.
2. User selects brewing temperature (or it defaults).
   Control app calls **BrewTemp.SetValue(...)** to set the brew thermostat to the chosen temperature.
3. If necessary, control app calls **Status.GetValue(...)** to get machine status. Or control app may already know the status from receiving **Status.PropertyChanged** notifications.
4. Water heats up. When machine status=3 (ready to brew), control app displays **ready-to-brew** message.
5. User clicks a **BREW** button on the control app screen.
6. Control app calls **Brew.SetValue(...)** to set the brew switch value to 2 (on).
7. Microcontroller sets **Status.Value** to **4 (brewing)**.
8. Pump starts. Brewing proceeds. Duration depends on cup size selected in step 1.
9. When brewing is done, the microcontroller turns the pump off and:
   - sets **Brew.Value** to **0 (off)**.
   - sets **Status.Value** to **3 (ready to brew)**.
10. **Status** object sends a **PropertyChanged** notification to the control app to report its value change.
11. On receiving this notification, the control app displays appropriate user messages, e.g. **"Bon Appetito".**

Jeff Berryman
Bosch Communications Systems
ja.Berryman@us.bosch.com
+1 952 457 5445
US East Coast