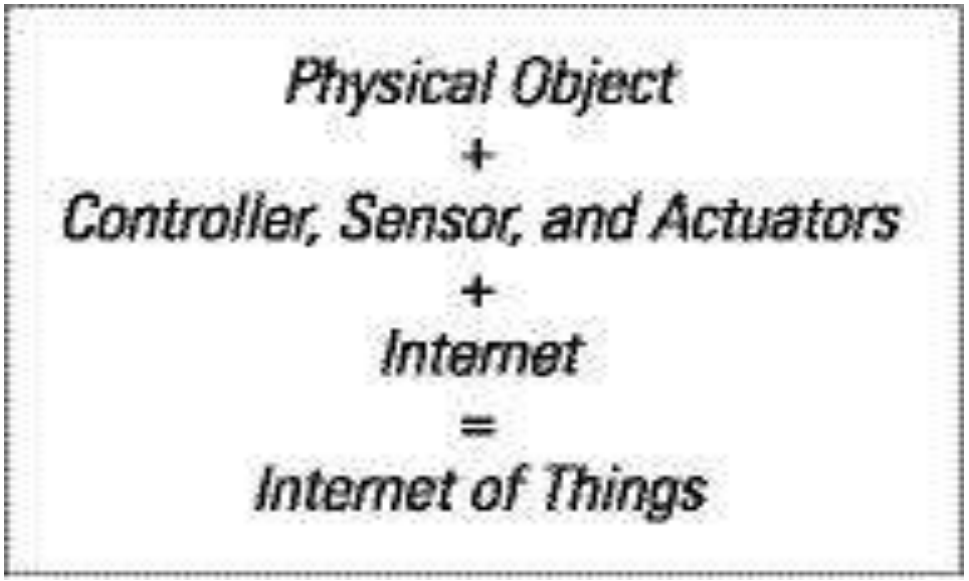
PROTOTYPING ONLINE COMPONENTS

- Internet of Things devices are **“magical” objects**: a physical thing with an embedded controller and sensors that allow it to do clever things that mechanical object couldn’t.
- Ex: Air fresheners that pump fragrance into the room only when they sense that someone has walked past.

- The controller and associated electronics allow it to sense and act on the real world.
- The Internet adds a dimension of communication.
- The network allows the device to inform you or others about events or to gather data and let you act on it in real time.
- It lets you aggregate information from disparate locations and types of sensors.
- Similarly, it extends your reach, so you can control or activate things from afar.

The key components of the Internet of Things



- Sensor devices which record temperature might write that data to Xively.
- **Xively** is an Internet of Things (IoT) platform owned by Google.
- Xively offers product companies a way to connect products, manage connected devices and the data they produce, and integrate that data into other systems.
- Notification devices like Bubblino blow bubbles in response to tweets on Twitter.
- Things should be "first class citizens" of the Internet, they do seem to be currently tied to particular websites or services.

- Each device is tied to a single web service.
- Although you've looked at existing services (Xively, Twitter), you might benefit from creating your own.

- For a personal project, creating such a service may not be important, but if you're developing a product to sell, you will want to be in control of the service—otherwise, you may have to recall every device to reprogram any time it is discontinued, changes terms and conditions, making your use of it abusive, or changes its API, making your code stop working.
- In fact, even Bubblino runs via a service at http://bubblino.com which allows users to customise their Bubblino to search for particular words.

GETTING STARTED WITH AN API

- The most important part of a web service, with regards to an Internet of Things device, is the Application Programming Interface, or API.
- An API is a way of accessing a service that is targeted at machines rather than people.
- If you think about your experience of accessing an Internet service, you might follow a number of steps.

Ex: **Flickr** is an American image hosting and video hosting service,

Flickr enables users to post photos from nearly any camera phone or directly from a PC.

For example, to look at a friend's photo on Flickr, you might do the following:

1. Launch Chrome, Safari, or Internet Explorer.
2. Search for the Flickr website in Google and click on the link.
3. Type in your username and password and click "Login".
4. Look at the page and click on the "Contacts" link.
5. Click on a few more links to page through the list of contacts till you see the one you want.
6. Scroll down the page, looking for the photo you want, and then click on it.

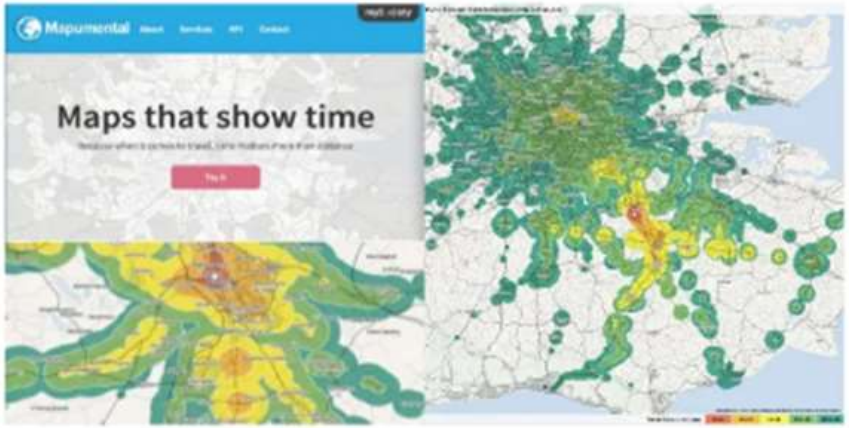
- These actions are simple for a human, but they involve a lot of looking, thinking, typing, and clicking.
- A computer can’t look and think in the same way.
- The tricky and lengthy process of following a sequence of actions and responding to each page is likely to fail the moment that Flickr slightly changes its user interface.
- For example, if Flickr rewords “Login” to “Sign in”, or “Contacts” to “Friends”, a human being would very likely not even notice, but a typical computer program would completely fail.
- Instead, a computer can very happily call defined commands such as login or get picture #142857.

MASHING UP APIS

- The data you want is already available on the Internet but in a form that doesn't work for you?
- The idea of "mashing up" multiple APIs to get a result has taken off and can be used to powerful effect.

- Using a mapping API to plot properties to rent or buy
- Ex:
  - Google Maps to visualise properties to rent via Craigslist, or Foxtons in London showing its properties using Mapumental.
  - Foxtons become first property player to use Mapumental maps on its website.

- **Mapumental** was a web-based application for displaying journeys in terms of how long they take, rather than by distance, a technique also known as isochrone or geospatial mapping.



– Showing Twitter trends on a global map or in a timeline or a charting API.

– Fetching Flickr images that are related to the top headlines retrieved from *The Guardian newspaper's API.*

The Guardian newspaper's API

- *This* API stores all articles, images, audio and videos dating back to 1999.
- All accessible with a single open platform key

- Some of the more visible and easy-to-use APIs want to embed your data within them—for example, the Google Maps API.
- This means that they are ideal to use within a web browser, but you aren’t in control of the final product, and there might be limited scope for accessing them from a microcontroller.

SCRAPING

- In many cases, companies or institutions have access to fantastic data but don’t want to or don’t have the resources or knowledge to make them available as an API.
- Screen-scraping can be used.

Screen scraping

- is the process of copying information that shows on a digital display so it can be used for another purpose.
- Visual data can be collected as raw text from on-screen elements such as a text or images that appear on the desktop, in an application or on a website.

Examples

- Adrian has scraped the Ship AIS system (www.shipais.com/, whose data is semi-manually plotted by shipping enthusiasts) to get data about ships on the river Mersey, and this information is then tweeted by the @merseyshipping account (www.mcqn.com/weblog/ connecting_river_mersey_twitter).
- The Public Whip website (www.publicwhip.org.uk/) is made possible by using a scraper to read the Hansard transcripts of UK government sessions (released as Word documents).

- As well as other tools for working with data online, the ScraperWiki site (https://scraperwiki.com) has an excellent platform for writing scrapers, in a number of dynamic programming languages, which collate data into database tables.

LEGALITIES

- Screen-scraping may break the terms and conditions of a website.
- For example, Google doesn't allow you to screen-scrape its search pages but does provide an API.
- Even if you don't think about legal sanctions, breaking the terms and conditions for a company like Google might lead to its denying you its other services, which would be at the very least inconvenient.

- Other data is protected by copyright or, for example, database rights.
- In one project a scraper is used that read football fixtures and moved a “compass” to point to the relative direction that team was playing in.
- However, certainly in the UK, fixtures lists are copyrighted, and the English and Scottish football leagues have legal proceedings against variousoperators for not paying them a licensing fee for that data (http://www. out-law.com/page-10985).
- For a personal pet project, creating such a scraper houldn’t be a huge issue but might reduce the viability of a commercial product (depending on whether the licensing costs are sensible business costs or prohibitive).

- Alternative sources of information often are available.
- For example, you could use OpenStreetMap instead of Google Maps.
- The UK postcode database is under Crown opyright, but there are other, perhaps partial, crowdsourced versions.

PROTOTYPIN ONLINE COMPONENTS

- WRITING A NEW API :
  - Clockodillo
  - Security
  - Implementing the API
  - Using cURL to Test
  - Going Further
- REAL-TIME REACTIONS
  - Polling
  - Comet
- OTHER PROTOCOLS
  - MQ Telemetry Transport
  - Extensible messaging and presence protocol
  - Constrained Application Protocol

WRITING A NEW API

- When you know what data you have, what actions can be taken on it, and what data will be returned, the flows of your application become simple.
- This is a great opportunity to think about programming without worrying (at first) about the user interface or interactions.
- Although this might sound very different from writing a web application, it is actually an ideal way to start: by separating the business problem from the front end, you decouple the model (core data structure) from the view (HTML/JavaScript) and controller (widgets, form interaction, and so on).
- If you've programmed in one of the popular MVC frameworks (Ruby on Rails, Django, Catalyst, and so on), you already know the advantage of this approach.
- The best news is, if you start designing an API in this way, you can easily add a website afterwards.

WRITING A NEW API : Clockodillo

- Clockodillo is an Internet-connected task timer.
- The user can set a dial to a number of minutes, and the timer ticks down until completed.
- It also sends messages to an API server to let it know that a task has been started, completed, or cancelled.
- A number of API interactions deal precisely with those features of the physical device:
  - Start a new timer
  - Change the duration of an existing timer
  - Mark a timer completed
  - Cancel a timer
  - View and edit the timer's name/description
- And, naturally, the user may want to be able to see historical data:
  - Previous timers, in a list
  - Their name/description
  - Their total time and whether they were cancelled

WRITING A NEW API : Security

- For Clockodillo , perhaps a boss might want to double-check that employees are using the timer.
- Or a competitor might want to check the descriptions of tasks to spy what your company is working on. Or a (more disreputable) competitor might want to disrupt and discredit the service by entering fake data.

Task	Inputs	Outputs
1. Create a new timed task	User, Timer duration	Timer ID
2. Change duration of timed task	User, Timer ID, New duration	OK
3. Mark timer complete	User, Timer ID	OK
4. Cancel timer	User, Timer ID	OK
5. Describe the timed task	User, Timer ID, Description	OK
6. Get list of timers	User	List of Timer IDs
7. Get information about a timer	User, Timer ID	Description, Create time, Status

WRITING A NEW API : Security

- For Clockodillo , you have to consider the risks in sending the identification or authentication data over the Internet—whether that's a MAC address or username and password.
- The two main cases here are as follows:
  - Someone who is targeting a specific user and has access to that person's wired or (unencrypted) wireless network. This attacker could read the details and use them (to create fake timers or get information about the user).
  - Someone who has access to one of the intermediate nodes. This person won't be targeting a specific device but may be looking to see what unencrypted data passes by, to see what will be a tempting target.
- If your device becomes popular, it may have competitors who would be delighted to expose a security flaw.

WRITING A NEW API : Security

- For a web API, you can simply do this by targeting https:// instead of http://. It doesn't require any further changes to your application code. It is easy to set up most web servers to serve HTTPS .
- The OAuth 1.0 protocol—used by services such as Twitter to allow third party applications to access your account without requiring your password— is a good example of providing strong authentication without using HTTPS.
- The content of the API call is still sent in the clear, so an attacker sniffing the network traffic would still be able to see what was happening, but he wouldn't be able to modify or replay the requests.

WRITING A NEW API : Implementing the API

- An API defines the messages that are sent from client to server and from server to client.
- Ultimately, you can send data in whatever format you want, but it is almost always better to use an existing standard because convenient libraries will exist for both client and server to produce and understand the required messages.
- Here are a few of the most common standards that you should consider:
- **Representational State Transfer (REST):** Access a set of web URLs using HTTP methods such as GET and POST, but also PUT and DELETE. The result is often XML or JSON but can often depend on the HTTP content-type negotiation mechanisms.
- **JSON-RPC:** Access a single web URL
- **XML-RPC:** This standard is just like JSON-RPC but uses XML instead of JSON.
- **Simple Object Access Protocol (SOAP):** This standard uses XML for transport like XML-RPC but provides additional layers of functionality, which may be useful for very complicated systems.

WRITING A NEW API : USING CURL TO TEST

- Clients for URLs is a project comprised of two development efforts- cURL and libcurl . Libcurl if free , client side URL transfer library with support of wide range of protocols . Curl is command tool for getting or sending files using URL syntax .
- While you are developing the API, and afterwards , to test it and show it off , you need a way to interact with it .
- curl simply makes an HTTP request and prints out the result to a terminal.

WRITING A NEW API : GETTING FURTHER

- API Rate Limiting : If the service becomes popular, managing the number of connections to the site becomes critical. Setting a maximum number of calls per day or per hour or per minute might be useful. You could do this by setting a counter for each period that you want to limit.
- While a software application can easily warn users that their usage limit has been exceeded and they should try later .
- OAuth for Authenticating with Other Services : While OAuth may not (currently) be the best solution for connecting with a microcontroller (at present, there are no accepted libraries for Arduino), there is no reason why the back-end service should not accept OAuth to allow hooks to services like Twitter, music discovery site last.fm, or the web automation of If This Then That.
- Interaction via HTML : The API currently serializes the output only in JSON, XML, and Text formats. You might also want to connect from a web browser.

WRITING A NEW API :GETTING FURTHER

- Drawbacks : Although web browsers do speak HTTP, they don't commonly communicate in all the methods that we've discussed. In particular, they tend to support the following:
- **GET:** Used to open pages and click on links to other pages.
- **POST:** Used when submitting a form or to upload files. To post a timer, you could create a web form like the following:

```
<form method="POST" action="/timers.html">
<input type="text" name="duration">
<input type="submit" value="Create a new timer!">
</form>
```

- This form calls the POST action and returns the appropriate HTML
- .

WRITING A NEW API :GETTING FURTHER

- Designing a Web Application for Humans
- For example, the following figure shows a static login page, to be served by GET. The API didn't even specify a GET action, as it was superfluous for a computer. This page is entirely for the convenience of a human.
- All the labels like "Your email address" and the help text like "Remember your password is case sensitive" are purely there to guide the user.
- The logo, as well as proving that we are really not designers, is there as a branding and visual look and feel for the site.

The human-facing Clockodillo login page.

WRITING A NEW API :GETTING FURTHER

- That's a simple example, but the following figure shows an even more extreme change. The list of timers, instead of being a JSON string containing a raw data structure, is highly formatted. The dates are formatted for humans. The duration of the timer and the status (in progress, completed, abandoned) are visualized with colors, progress bars, and a duration "badge".
- The page also links to other actions: The "Edit" button opens a page that allows access to the actions that change description, and so on. The menu bar at the top links to other functions to help users flow through the tasks they want to carry out.
- Looking at your product from both contrasting perspectives (machine and human) will make it stronger and better designed.

The human-facing list of timers.

REAL-TIME REACTIONS

- For a bare-bones board such as the Arduino, the current Ethernet/HTTP shields and libraries tend to block during the connection, which means that during that time, the microcontroller\ can't easily do any other processing .
- If you want to perform an action the instant that something happens on your board, you may have to factor in the connection time.
- If the server has to perform an action immediately, that "immediately" could be nearly a minute later, depending on the connection time.
- We look at two options here: polling and the so-called "Comet" technologies.

REAL-TIME REACTIONS : polling

- If you want the device or another client to respond immediately, how do you do that? You don't know when the event you want to respond to will happen, so you can't make the request to coincide with the data becoming available.
- Consider these two cases:
  - The Where Dial should start to turn to "Work" the moment that the user has checked into his office.
  - The moment that the task timer starts, the client on the user's computer should respond, offering the opportunity to type a description of the task.

REAL-TIME REACTIONS : polling

- The traditional way of handling this situation using HTTP API requests was to make requests at regular intervals. This is called polling.
- You might make a call every minute to check whether new data is available for you.
- However, this means that you can't start to respond until the poll returns. So this might mean a delay of (in this example) one minute plus the time to establish the HTTP connection.
- You could make this quicker, polling every 10 seconds, for example.
- But this would put load on the following:
  - **The server:** If the device takes off, and there are thousands of devices, each of them polling regularly, you will have to scale up to that load.
  - **The client:** This is especially important if, as per the earlier Arduino example, the microcontroller blocks during each connect!

REAL-TIME REACTIONS : Comet

- Comet is an umbrella name for a set of technologies developed to get around the inefficiencies of polling.
- Long Polling (Unidirectional) :
- The first important development was “long polling”, which starts off with the client making a polling request as usual.
- However, unlike a normal poll request, in which the server immediately responds with an answer, even if that answer is “nothing to report”, the long poll waits until there is something to say.
- This means that the server must regularly send a keep-alive to the client to prevent the Internet of Things device or web page from concluding that the server has simply timed out.

REAL-TIME REACTIONS : Comet

- Long Polling (Unidirectional) where it can be used :
- Long polling would be ideal for the case of **WhereDial:** the dial requests to know when the next change of a user's location will be. As soon as WhereDial receives the request, it moves the dial and issues a new long poll request.
- Of course, if the connection drops (for example, if the server stops sending keep-alive messages), the client can also make a new request.

- However, it isn't ideal for the task timer, with which you may want to send messages from the timer quickly, as well as receive them from the server. Although you can send a message, you have to establish a connection to do so.

REAL-TIME REACTIONS : Comet

- Multipart XMLHttpRequest (MXHR) (Unidirectional) :
- When building web applications, it is common to use a JavaScript API called XMLHttpRequest to communicate with the web server without requiring a full new page load.
- Note that XMLHttpRequest is a misnomer because there's no requirement to actually use XML at all. Using this content type is perhaps more sophisticated if you want to be able to receive multiple messages from the server.
- In the example of **WhereDial**, this is unlikely; you're unlikely to change location first to Home and then to Work in quick succession.
- However, for an Internet of Things device such as **Adrian's Xively meter**, which tries to show the state of a Xively feed in real time, being able to respond to changes from the server almost immediately is the essential purpose of the device.

REAL-TIME REACTIONS : Comet

- HTML5 WebSockets (Bidirectional) :
- Traditionally, the API used to talk directly to the TCP layery is known as the sockets API. When the web community was looking to provide similar capabilities at the HTTP layer, they called the solution WebSockets.
- WebSockets have the benefit of being bidirectional.
- You can consider them like a full Unix socket handle that the client can write requests to and read responses from.
- This might well be the ideal technology for the task timer. After a socket is established, the timer can simply send information down it about tasks being started, modified, or cancelled, and can read information about changes made in software, too.

OTHER PROTOCOLS

- MQ TELEMETRY TRANSPORT :
- MQTT is a lightweight messaging protocol, designed specifically for scenarios where network bandwidth is limited or a small code footprint is desired.
- It was developed initially by IBM but has since been published as an open standard, and a number of implementations, both open and closed source, are available, together with libraries for many different languages.

- Rather than the client/server model of HTTP, MQTT uses a publish/ subscribe mechanism for exchanging messages via a message broker.

- Rather than send messages to a pre-defined set of recipients, senders publish messages to a specific topic on the message broker.

OTHER PROTOCOLS

- **MQ** TELEMETRY TRANSPORT :

- Recipients subscribe to whichever topics interest them, and whenever a new message is published on that topic, the message broker delivers it to all interested recipients.

- This makes it much easier to do one-to-many messaging, and also breaks the tight coupling between the client and server that exists in HTTP.

- A sister protocol, MQTT for Sensors **(MQTT-S),** is also available for extremely constrained platforms or networks where TCP isn't available, allowing MQTT's reach to extend to sensor networks such as ZigBee.

OTHER PROTOCOLS

- **EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL :**

- Another messaging solution is the Extensible Messaging and Presence Protocol, or **XMPP**.

- **XMPP** grew from the Jabber instant messaging system and so has broad support as a general protocol on the Internet.

- This is both a blessing and a curse: it is well understood and widely deployed, but because it wasn't designed explicitly for use in embedded applications, it uses **XML** to format the messages.

- This choice of **XML** makes the messaging relatively verbose, which could preclude it as an option for **RAM**-constrained microcontrollers.

OTHER PROTOCOLS

- **CONSTRAINED APPLICATION PROTOCOL :**

- The Constrained Application Protocol (CoAP) is designed to solve the same classes of problems as HTTP but, like MQTT-S, for networks without TCP.

- There are proposals for running CoAP over UDP, SMS mobile phone messaging, and integration with 6LoWPAN.

- CoAP draws many of its design features from HTTP and has a defined mechanism to proxies to allow mapping from one protocol to the other.

- At the time of this writing, the protocol is going through final stages of becoming a defined standard, with the work being coordinated by the Internet Engineering Task Force Constrained RESTful Environments Working Group.

PROTOTYPIN ONLINE COMPONENTS (part 2)

- ✓ WRITING A NEW API :
  - Clockodillo
  - Security
  - Implementing the API
  - Using cURL to Test
  - Going Further
- ✓ REAL-TIME REACTIONS
  - Polling
  - Comet
- ✓ OTHER PROTOCOLS
  - MQ Telemetry Transport
  - Extensible messaging and presence protocol
  - Constrained Application Protocol