Prototyping Connected Devices for the Internet of Things

Keywords

Internet of Things, connected devices, rapid prototyping, representational state transfer (REST), machine-to-machine, web services, cloud computing, Microsoft .NET Gadgeteer.

Abstract

The vision of the "Internet of Things" is one where the explosion in connectivity we have seen over the last decade increasingly extends right down to the simplest of electronic devices – to the point where pretty-much any thing may connect to the internet. There is tremendous potential in this vision but there are of course also a great many complex technical, social and economic questions which are yet to be addressed. We firmly believe that the ability to quickly prototype, test and deploy real devices will be a key element in accelerating our understanding of these challenges and of the benefits of networked things.

In this paper we outline some of the hardware and software tools and platforms currently available which can be used to facilitate the creation of networked embedded devices. In particular we illustrate the possibilities these afford by focussing on a specific platform – Microsoft .NET Gadgeteer – which we describe in some detail and illustrate using a series of examples. Key elements of tools like Gadgeteer include: rapid construction and reconfiguration of electronic device hardware; ease of programming and debugging; and the ability to leverage online web services for additional storage, communication and processing capabilities. We hope that practitioners in this exciting field will be able to build on the experiences and examples reported here as the Internet of Things becomes ever-closer to reality.

Introduction

Today, devices such as personal computers and smartphones form a significant fraction of internet-connected devices globally [4]. In the coming decade, analysts predict that simpler embedded devices will increasingly complement these established platforms as peers on the internet in a growing machine-to-machine communication paradigm [3], [4], [9]. In addition to networked versions of devices which are commonplace today – things like washing machines, alarm clocks, doorbells and light switches – pundits are predicting completely new applications. They imagine devices around us that continuously communicate with each other and improve our productivity at work, help us manage our daily activities, let us keep in touch with others more easily, provide information in a timely and convenient fashion, and enhance our leisure time. Some estimates predict that this "Internet of Things" will constitute 100 billion devices as soon as 2020 [3], [9].

One particular shift we expect to see is that the software running on the processors of embedded devices will increasingly be complemented by cloud-based web services, made accessible via built-in network interfaces which leverage established web-based protocols such as HTTP and XML. Web services will dramatically extend the effective processing and storage abilities of the connected devices which comprise the Internet of Things, enabling relatively cheap embedded processors to leverage sophisticated data processing and gain access to large datasets. Ultimately a new class of

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE

applications may emerge, where groups of devices act as the input and output elements of potentially global-scale distributed services and applications.

With so many possibilities across the broad realm of the Internet of Things, tools which expedite the reduction of ideas to working prototypes may have an important role to play. By prototyping and deploying live systems early on in the concept development cycle it is possible to understand the strengths and weaknesses of a particular application, design or specific implementation sooner and feed this information back into an iterative development process. Many tools also make it possible for individuals with little experience of connected device development to realise working prototypes thereby extending accessibility to a great many developers, designers, researchers and enthusiasts. We believe that the combined engagement of this spectrum of creative users is likely to accelerate the speed with which networked devices are adopted simply because of the number of application scenarios which can be explored, developed and deployed.

In this paper we describe some of the tools and services which are currently available to support the rapid development of networked devices and we drill down on one particular system, Microsoft .NET Gadgeteer (http://netmf.com/gadgeteer, hereafter simply 'Gadgeteer'). Gadgeteer is a general-purpose device development platform which has been reported previously in the literature [11]. In this paper we recap its main features and for the first time present details relating to the networking support which enables the platform to be used for the rapid prototyping of connected devices for the Internet of Things. We also introduce Gadgeteer's ability to leverage online storage and processing by way of complete examples which we hope other practitioners will be able to replicate and build upon. The paper ends by summarising some of the pros and cons of different connected device prototyping tools and the possibilities these afford for scaling up to larger numbers of devices.

Tools for prototyping connected devices and systems

A number of tools for developing machine-to-machine communication concepts into working systems are becoming well established. One popular tool is the Arduino platform (http://arduino.cc), a family of embedded processors that can be programmed using the C language via an accessible and minimalist integrated development environment (IDE). Debugging with Arduino is typically supported via simple communications over a serial line interface. In terms of electronic hardware, Arduino processors are complemented by an ecosystem of 'shields' – add-on circuit boards that extend the basic capabilities of Arduino (http://shieldlist.org/).

From a hardware perspective, the use of Arduino for connected device development is enabled by the availability of shields that provide Ethernet, WiFi or GPRS connectivity, see Figure 1 for example. On the software side, a technique known as representational state transfer (REST) [8] is commonly used as a lightweight and easy-to-debug mechanism for communication between connected devices such as those built with Arduino. REST inter-device communication is enabled via services which are exposed and accessed using HTTP and is readily supported by Arduino libraries that implement the relevant networking protocols and enable simple webserver operation. The widespread use of Arduino means that there is also a vibrant community of users who create, share and support their own libraries and examples online, further facilitating the development of new applications.

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE

The mbed programmable microcontroller platform (http://mbed.org) is another embedded electronics development platform which is growing in popularity. There are currently two different mbed microcontroller products which take the form of small rectangular modules with protruding pins which allow the device to be inserted into a breadboard during prototyping, see Figure 1. This form factor also allows the module to be integrated into a custom printed circuit board (PCB) should that subsequently be necessary. A key difference from Arduino is the mbed online IDE, which is accessible via a web browser without the need to install any software. Extensive documentation and libraries are available through the IDE which also supports the sharing of user-generated code samples and libraries. In support of connected device development, one of the mbed variants includes built-in Ethernet connectivity and the mbed code repository includes a comprehensive set of networking libraries and examples. Support for debugging code running on the mbed has been the limited to date, but it is possible to transition to a more traditional PC-based IDE if necessary and an upcoming version of mbed aims to provide better support for debugging via the online IDE.

In addition to microcontroller-based platforms such as Arduino and mbed, there are also a large number of small-form factor devices that run Linux. These bring with them the opportunity to leverage an extensive set of pre-existing tools and to reuse existing software components such as Node.js (http://nodejs.org/) which simplifies the implementation of REST-like asynchronous web-based APIs. These platforms are powerful and flexible but as a result they can expose more complexity to the user and they are typically less cost effective than Arduino for lightweight device development. However, new products with very low price points such as Raspberry Pi (http://www.raspberrypi.org/) and the BeagleBone (http://beagleboard.org/bone) are becoming extremely popular and as a result have active and growing online communities.

One final system we want to highlight and which we will use in the rest of this paper to illustrate how simple it can be to prototype connected devices is Microsoft .NET Gadgeteer. This is a modular platform designed to facilitate the construction of prototype digital devices [11]. A central 'mainboard' containing a CPU and a number of sockets may be connected to a growing number of commercially available 'modules' – different sensors, actuators, displays, communication and storage elements. The solder-less composability of hardware components allows prototypes to be constructed, re-configured and extended very quickly. The Gadgeteer system is tightly integrated with the Microsoft Visual Studio IDE which provides a great deal of support throughout the prototyping process. The Intellisense system performs dynamic syntax checking and continually provides hints and prompts to ease coding. The IDE also aids debugging via breakpoints, single stepping, variable watches and execution traces.

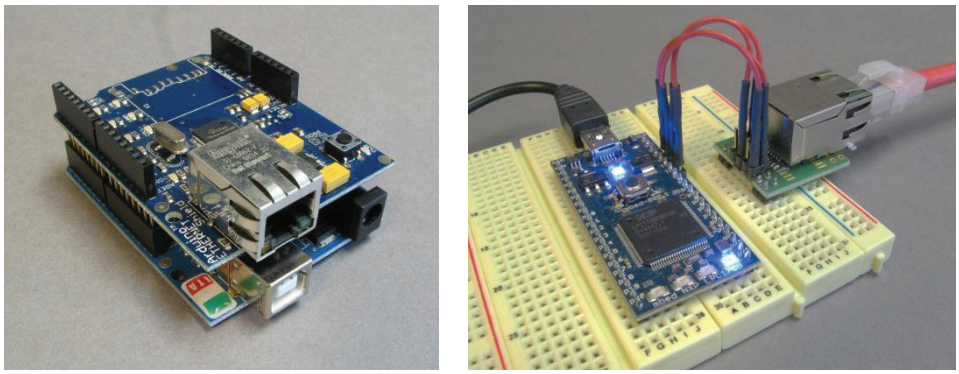


Figure 1. The Arduino device prototyping platform with Ethernet shield (left) and the mbed embedded development platform (right).

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE

Microsoft .NET Gadgeteer design choices

The primary design goal of Gadgeteer was to simplify the development of applications as much as possible, even if this is at the expense of performance. Device functionality is programmed using C# or Visual Basic; the use of managed code is unusual for embedded device development where C-like languages are firmly established, but our experience shows that it tends to reduce the time and expertise needed for prototyping new applications [10]. In a pre-production environment this offsets the increased processor and memory requirements. In a similar way, Gadgeteer uses an event-based model wherever possible which further simplifies the creation of many applications and helps developers familiar with event-based programming on desktop and mobile platforms to transition to embedded device development.

The functionality of each physical Gadgeteer module is encapsulated in a software library through an intuitive high-level application programming interface (API). The high level of abstraction often allows modules to be used in quite sophisticated ways with just a few lines of code, enabling users with relatively little experience to build compelling devices and applications. This approach lowers the barrier to entry, but doesn't limit the flexibility available to more experienced developers. If a different abstraction or functionality is required it is possible to build on top of lower-level APIs to encapsulate this.

The .NET Micro Framework (http://netmf.com) which underpins Gadgeteer contains extensive provision for networking. The Gadgeteer networking API builds on top of this in a way which supports a compact and easy to understand design pattern for responding to REST-ful web requests with text, images or byte streams. Supporting this was prioritised above other web-related functionality such as serving a hierarchy of content in the manner of a traditional web server in order to simplify Internet of Things application development.

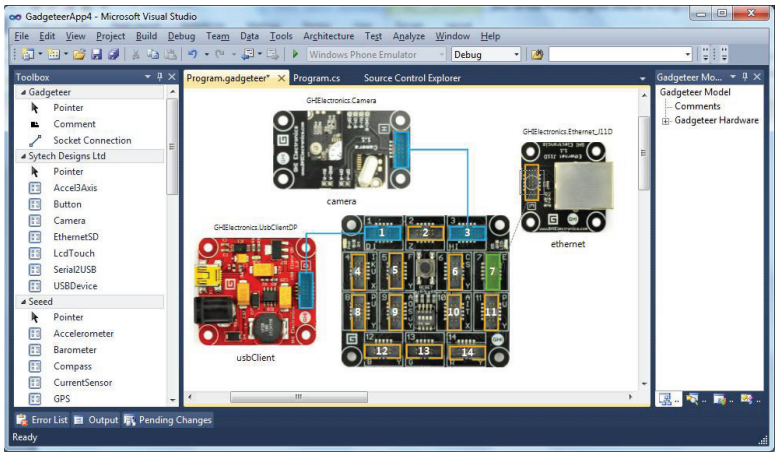
Building a simple web-connected device with Gadgeteer

To illustrate how straightforward it can be to create a connected device with a REST-ful interface, Figure 2 presents a simple "internet webcam" built using Gadgeteer. The process of creating a device starts with a graphical design tool, shown in Figure 2(a), which allows the developer to specify the hardware components they want to use and how they can be connected to a mainboard. Having 'wired these up' graphically on-screen, it takes just a couple of minutes to construct the corresponding physical hardware, Figure 2(b). In this example, the webcam consists of a Gadgeteer mainboard connected to Ethernet, camera and power supply modules. The code required to encapsulate the hardware configuration is automatically generated and the appropriate libraries are linked in.

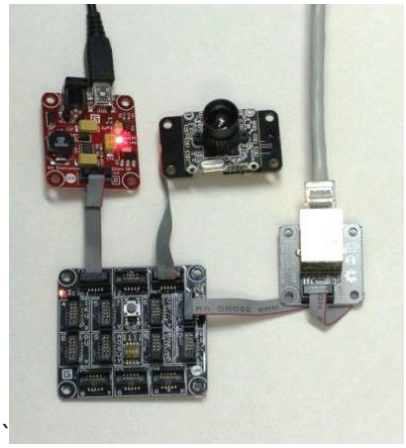
With Gadgeteer, a web server can be set up using a single line of code once a network connection has been established. Gadgeteer's event-based style lends itself to handling REST-ful requests through the creation of event handlers for each desired HTTP request path; each handler simply responds to the associated incoming request with the appropriate object. Strings (including complete HTML pages), images and data streams are supported directly by the Gadgeteer API. In terms of the webcam application, the capture of a new webcam image is triggered remotely via an HTTP request to a web server running on the device. At the same time, the most recently captured image is returned to the web client initiating the request. Figure 2(c) lists the C# code required to

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE

implement the necessary functionality – just twelve lines of code excluding the auto-generated function prototypes. Of course, a more robust implementation would cover a variety of potential error conditions, such as lack of network connectivity, but the simplest implementation is shown here for clarity. The web browser screenshot in Figure 3 shows the device in operation.



(a)



(b)

```
GT.Picture lastPicture;
GT.Networking.WebEvent cameraServer;

void ProgramStarted()
{
        // associate PictureCaptured event with its handler
        camera.PictureCaptured += new Camera.PictureCapturedEventHandler(camera_PictureCaptured);

        // request DHCP address and associate handler for network setup
        ethernet.UseDHCP();
        ethernet.NetworkUp += new GTM.Module.NetworkModule.NetworkEventHandler(ethernet_NetworkUp);
}

void ethernet_NetworkUp(GTM.Module.NetworkModule sender,
                                GTM.Module.NetworkModule.NetworkState state)
{
        // start a webserver on port 80
        WebServer.StartLocalServer(ethernet.NetworkSettings.IPAddress, 80);

        // set up a handler for http '/picture' requests
        cameraServer = WebServer.SetupWebEvent("picture");
        cameraServer.WebEventReceived += new
                                WebEvent.ReceivedWebEventHandler(cameraServer_WebEventReceived);

        // capture an initial picture
        camera.TakePicture();
}

void cameraServer_WebEventReceived(string path, WebServer.HttpMethod method, Responder responder)
{
        // return the last picture and initiate a new picture
        responder.Respond(lastPicture);
        camera.TakePicture();
}

void camera_PictureCaptured(Camera sender, GT.Picture picture)
{
        lastPicture = picture;
}
```

(c)

Figure 2. An "internet webcam" constructed from .NET Gadgeteer. (a) The hardware configuration which includes an RJ45 module for a wired Ethernet connection is entered graphically in Visual

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE

Studio. When a module connector is selected, compatible mainboard sockets are highlighted in green to aid the user in wiring up their design. (b) A photo of the corresponding physical hardware. (c) Just 12 lines of code are enough to create a webserver which responds to incoming requests with the most recent image and simultaneously triggers the capture of a new image capture. Note that all the function definitions in this example are automatically generated by the Visual Studio IDE.

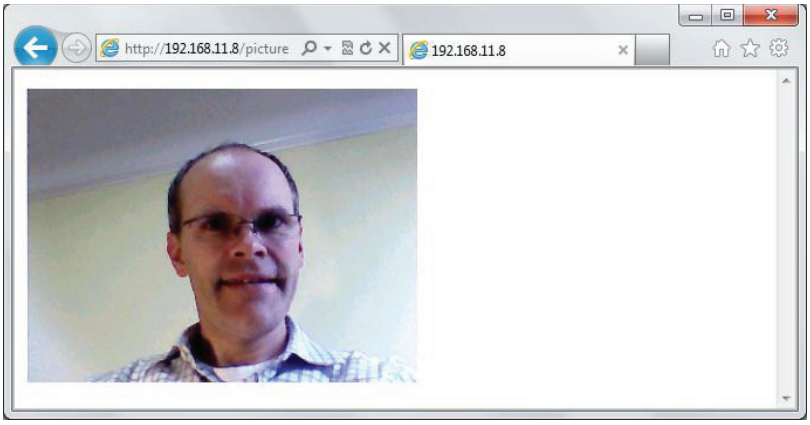
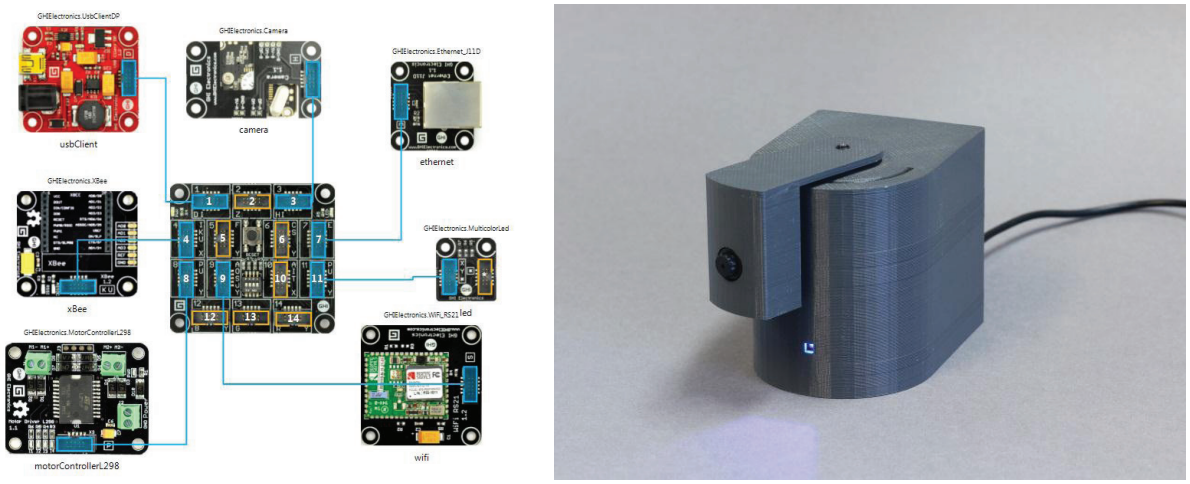


Figure 3. A web request to the network-connected camera device returns the previous picture and also initiates the capture of a new image.

A more sophisticated web-controlled camera is shown in Figure 4. In this case the camera module is attached to a servo motor controlled arm, allowing remote panning as well as image capture, again over a REST-ful interface. In addition to a wired Ethernet connection, this device also incorporates wireless 802.11 and Zigbee network interfaces. The former provides an alternative way of connecting to the internet, should a wired connection not be available or convenient. The latter was used in our prototype to provide an onward connection to lighter-weight Gadgeteer devices such as temperature and light-level sensors, effectively giving them a presence on the internet via additional software running on the camera device which acted as a bridge.



(a) (b)

Figure 4. A remote-controllable networked camera with servo-controller pan mechanism. In this case the electronic modules are housed in a 3D-printed plastic enclosure.

Storing, retrieving and sharing data

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE

Whilst an embedded webserver allows a device to expose state or functionality over HTTP, the ability to store, retrieve and share data is a key element of Internet of Things applications and for this an embedded web *client* API is equally important. For this reason, the Gadgeteer libraries were designed to ensure that making a web request is also straightforward; when the details of the HTTP request have been specified, an event handler is created to deal with the anticipated response and then the request itself is sent. In addition to supporting true peer-to-peer communication, in our experience with connected device development this approach is an intuitive way of providing access to a growing number of hosted web services that support the process of exchanging data between connected devices. These tools, which include cosm (formerly Pachube), Thingspeak and Nimbits (https://cosm.com/, https://thingspeak.com/ and http://www.nimbits.com/), make use of HTTP and XML to implement REST-ful APIs and are therefore readily accessible to platforms like Gadgeteer.

To give a practical example of this, Figure 5 lists the four lines of C# code needed to upload a barometric pressure reading to the online cosm repository. A complete connected device which continuously records and uploads sensor readings is shown in Figure 6, along with a screenshot of the cosm web interface for visualising the associated temperature and pressure data.

```
HttpRequest request = HttpHelper.CreateHttpPutRequest("http://api.pachube.com/v2/feeds/" + feedId +
    ".csv", PUTContent.CreateTextBasedContent(locationId + "temperature," +
    sensorData.Temperature.ToString() + "\n" + locationId + "pressure," +
    sensorData.Pressure.ToString()), "text/csv");
request.AddHeaderField("X-PachubeApiKey", apiKey);
request.ResponseReceived += new HttpRequest.ResponseHandler(req_ResponseReceived);
request.SendRequest();
```

Figure 5. A snippet showing the code required to upload a sensor reading to the cosm web service.

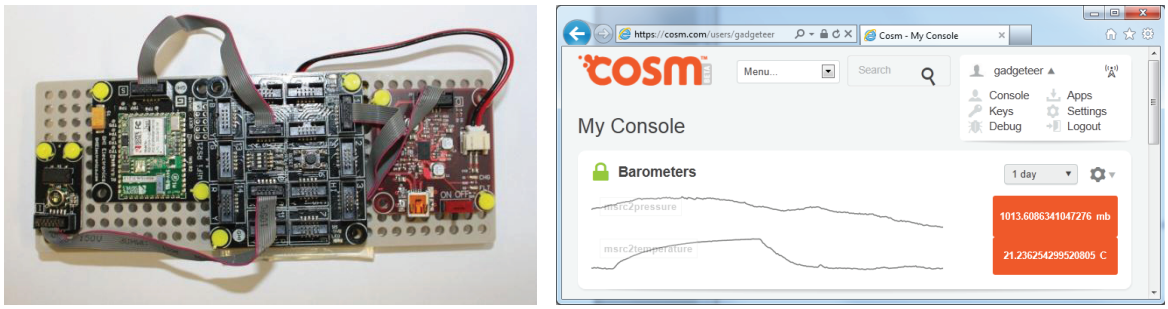


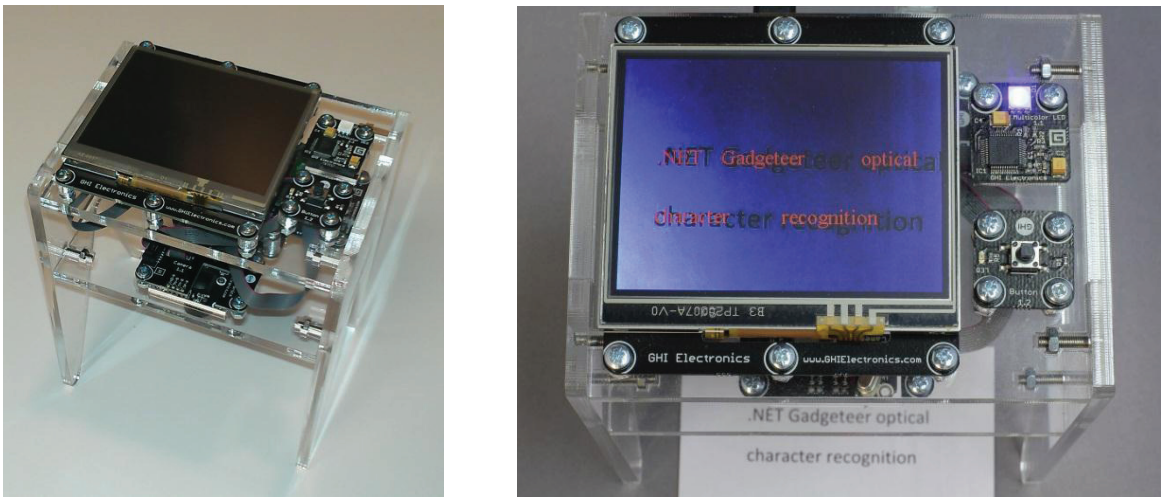
Figure 6. A Gadgeteer prototype which periodically stores temperature and pressure readings using the cosm web service via WiFi (left). A screenshot showing plots of data collected over a 24 hour period (right). Note that the prototype device is assembled using a perforated plastic baseboard which the Gadgeteer modules are attached to using yellow plastic pop-rivets.

Cloud-based processing for connected devices

In addition to communication between devices via online repositories such as cosm, one of the key benefits of connected operation is the potential to leverage cloud-based computation. Services such as Amazon EC2 and Microsoft Azure provide a mechanism to deploy online compute services which can be used to offload computation from connected devices. Project Hawaii from Microsoft Research (http://research.microsoft.com/hawaii/) is a ready-to-use web services test bed built on Azure which provides a variety of functionality free of charge for non-commercial applications.

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE

Hawaii currently provides off-the-shelf services to support certain computationally intensive processes in addition to basic communication between remote devices and online data storage.



(a) (b)

Figure 7. (a) The OCR device which has a camera facing down towards the worktop. (b) An image of some printed text which was captured and which has the OCR'ed text correctly overlaid in red care-of the Hawaii cloud-based OCR service.

To demonstrate how Hawaii can be used to extend the capabilities of Gadgeteer, Figure 7 shows another camera device we have prototyped. As with a traditional digital stills camera, an image is captured when the "shutter" button is pressed. At this point, rather than simply displaying the image and storing a copy locally, the image is sent to the Hawaii optical character recognition (OCR) service whereupon it is processed and any text detected is returned for display. Figure 8 shows the C# code which forms the basis of this example. When the picture to be sent to Hawaii for OCR processing has been captured and displayed, an HTTP request is created. This request incorporates the image, the appropriate authentication information and the necessary HTTP header fields. The resulting response from Hawaii triggers an event handler which needs to process the XML-enabled results. For simplicity, the code presented in Figure 8 simply selects the first word returned by the OCR service and includes no error handling. A more complete implementation would process all the text and associated meta-data as well as dealing with error conditions. The .NET Micro Framework includes native XML parsing and exception handling capabilities which make more complete and robust decoding simple to implement.

```
void ProgramStarted()
{
    ethernet.UseDHCP();
    button.ButtonPressed += new Button.ButtonEventHandler(button_ButtonPressed);
    camera.PictureCaptured += new Camera.PictureCapturedEventHandler(camera_PictureCaptured);
}

void button_ButtonPressed(Button sender, Button.ButtonState state)
{
    camera.TakePicture();
}

void camera_PictureCaptured(Camera sender, GT.Picture picture)
{
    // Show the picture on the display
    display.SimpleGraphics.DisplayImage(picture.MakeBitmap(), 0, 0);
```

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE

```
    // create and send an HTTP request which will send the picture to the Hawaii OCR service
    HttpRequest request = HttpHelper.CreateHttpPostRequest("http://157.55.188.73/OCR",
        POSTContent.CreateBinaryBasedContent(picture.PictureData), "image/jpeg");
    request.AddHeaderField("Authorization", "Basic " +
        ConvertBase64.ToBase64String(Encoding.UTF8.GetBytes("<insert your appID here>"));
    request.AddHeaderField("Cache-Control", "no-cache");
    request.ResponseReceived += new HttpRequest.ResponseHandler(request_ResponseReceived);
    request.SendRequest();
}

void request_ResponseReceived(HttpRequest sender, HttpResponse response)
{
    // for this example we just display the first OCR'ed word returned by Hawaii
    // by looking between the "<Text>" and "</Text>" tags
    int start = response.Text.IndexOf("<Text>", 0) + 6;
    int end = response.Text.IndexOf("</Text>", 0);
    display.SimpleGraphics.DisplayText(response.Text.Substring(start, end - start),
        Resources.GetFont(Resources.FontResources.NinaB), GT.Color.Red, 0, 0);
}
```

Figure 8. Using Gadgeteer to build an embedded device which leverages the Hawaii web services.

To further illustrate how Gadgeteer may be used to explore applications which leverage the ubiquitous connectivity which underpins the Internet of Things, we have built and deployed another camera-based application. The motivation behind this project was to replicate the work of Kuzuoka and Greenberg [5] who explored the use of telepresence proxies. These are devices which incorporate cameras and displays and are configured to share images between different physical locations. We built a number of networked Gadgeteer devices, each incorporating a display and camera, and developed a simple application which would periodically take a photo and upload it to an Azure-based web service such as the Hawaii Key-Value Store. The web service was configured to make the most recent photo from each device available to all the other telepresence devices. By displaying the latest photos from other devices in a round-robin sequence, a level of mutual awareness between users at different physical locations was maintained. By deploying these devices we were able to experience this lightweight form of telepresence and at the same time we explored different device form factors by creating a range of enclosures for the necessary electronics. Figure 9 shows some of the prototypes.

Figure 9. Using the flexible nature of the Gadgeteer hardware, we explored several different form factors for a network-connected telepresence device. An integrated unit containing all the modules is shown on the left, whilst the centre and right-hand prototypes have separate units for the display and camera. In the case of the right-hand design, the two units operate independently – each has its own Gadgeteer mainboard with a separate wired Ethernet connection.

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE

Selecting a platform for Internet of Things development

We have used Gadgeteer to illustrate how to build connected devices, but there are of course a great many factors to consider when choosing a development tool. For example, community support is a key element of any development platform and this is very much true of tools like Arduino, mbed and Gadgeteer. In each case online forums provide a mechanism for both new and experienced users to pose questions, exchange experiences and share code.

One of the premises behind the Gadgeteer software stack is that concise code empowers less experienced users to create useful applications and at the same time allows seasoned developers to build prototypes more quickly. This intuition is borne out through the anecdotal evidence we've collected at various Gadgeteer events; users report a very low hurdle for creating simple projects [10], yet have been able to build relatively sophisticated prototypes including connected devices [2]. Indeed, our experience shows that the simplicity of hardware and software development with Gadgeteer inspires students as young as 13 to engage with the platform [5], and this may ultimately prove valuable for educating a future generation of Internet of Things developers. Of course, other platforms will continue to be very relevant as well. For example professionals and hobbyists alike can leverage a growing set of samples and libraries for connecting a networked Arduino to online services, whilst educators have access to platforms like the Open University's SenseBoard (http://sense.open.ac.uk/) which has already been used to teach the Internet of Things concept as reported elsewhere in this Special Issue [6]. In the future it may even be possible to create devices and services in the realm of the Internet of Things using code-free development environments like Scratch (http://scratch.mit.edu/).

Key points of differentiation between the tools discussed in this paper include performance, debugging support, cost, power consumption and form factor. In the case of Gadgeteer, a high performance processor was chosen to support managed code and real-time debugging even though it results in a modest price increment over tools like Arduino and mbed. Power consumption was not an area of focus during the initial development of Gadgeteer but is a topic that we are currently exploring. Flexibility over form factor was a central design consideration for Gadgeteer and influenced the decision to use cables for interconnecting modules as opposed to a 'stacking' approach. This allows a variety of physical prototyping approaches, some of which have been illustrated in this paper.

No matter which tool or tools are used for prototyping, at some point it becomes necessary to build and deploy a greater number of devices, either for larger scale deployments or ultimately for mass-production. In our work so far we have used Gadgeteer for deployments of up to around 50 devices. This is relatively straightforward because of the ease of replication of a proven Gadgeteer design and the robustness of the assembled units. At some point it becomes more cost-effective to move to a custom PCB, which can be made more cheaply through circuit integration. Like several of the open hardware platforms, with Gadgeteer this process is facilitated by freely available hardware designs from many manufacturers.

More recently, tools specifically designed to facilitate the mass production of connected devices have started to emerge such as the ioBridge (http://www.iobridge.com/) and the Electric Imp (http://electricimp.com/). The latter includes a programmable processor and WiFi radio in a small package, which connects to applications through a hosted web service. In essence, these devices act

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE

as a single physical component which provides a link between a hardware interface and HTTP-based web APIs.

Conclusions

In this paper we demonstrated how rapid development tools can be used to prototype connected devices in the context of the growing Internet of Things. By combining rapid electronic hardware development with high-level software libraries and good debugging support, it is possible to build and iterate network-connected devices remarkably quickly, as illustrated by the Gadgeteer-based examples presented in this paper. We have also shown how web services like cosm and Hawaii can provide storage, communication and computation, enabling further application scenarios.

As the number of network-connected devices in the world continues to grow, it is clear that no single technology will prevail – the success of the Internet of Things is inherently tied to a heterogeneity of devices, protocols, services and applications. There are still many open questions within this broad scope and the tools presented here can't necessarily resolve these issues directly. However, as the Internet of Things vision gradually becomes a reality we believe that it will be important to explore the design space by way of working prototypes. We imagine these prototypes will be conceived and built by a diverse set of developers, researchers, designers and hobbyists and that tools like Gadgeteer will be valuable in this regard. We acknowledge that some applications will always be outside the scope of a rapid-prototyping toolkit but we nonetheless hope that others working in this exciting field will find these tools useful for exploring relevant issues and for quickly prototyping new ideas and applications, as we have done.

References

[1] Massimo Banzi, "Getting Started with Arduino". O'Reilly, 2008. ISBN: 978-0-596-15551-3
[2] Pollie Barden et al., "Telematic Dinner Party: Designing for Togetherness through Play and Performance", In Proceedings of Designing Interactive Systems, DIS 2012, June 2012
[3] Casaleggio Associati "The Evolution of Internet of Things", February 2011, http://www.casaleggio.it/pubblicazioni/Focus_internet_of_things_v1.81%20-%20eng.pdf
[4] John Gantz, "The Embedded Internet: Methodology and Findings", IDC, January 2009
[5] Steve Hodges et al., ".NET Gadgeteer: Experiences with a new platform for K-12 computer science education", to appear in Proceedings of the 44th SIGCSE Technical Symposium on Computer Science Education, March 2013
[6] Gerd Korteum et al., "Educating the Internet-of-Things Generation", IEEE Computer
[7] Hideaki Kuzuoka and Saul Greenberg. 1999. Mediating awareness and communication through digital but physical surrogates. In CHI '99 extended abstracts. pp. 11-12.
[8] Leonard Richardson and Sam Ruby, RESTful Web Services, O'Reilly, 2007.
[9] Constantine A. Valhouli, "The Internet of things: Networked objects and smart devices", The Hammersmith Group Research Report, February 2010
[10] N. Villar, J. Scott and S. Hodges. Prototyping with Microsoft .NET Gadgeteer. In Proceedings of the fifth International Conference on Tangible, Embedded and Embodied Interaction, TEI 2011.
[11] Nicolas Villar et al., ".NET Gadgeteer: A Platform for Custom Devices", in Proceedings of Pervasive 2012, Lecture Notes in Computer Science, June 2012

Acknowledgements

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE

The authors would like to acknowledge John Helmes for the industrial design of some of our prototypes, the Project Hawaii team behind the Hawaii web services described in this paper, and the extensive community of people who have contributed to the development of the .NET Gadgeteer prototyping system. We would also like to thank our reviewers for their valuable feedback.

Biographies

Steve Hodges is a Principal Hardware Engineer at Microsoft Research, Cambridge where he leads the Sensors and Devices research group. His research interests broadly fall into two categories: novel electronic devices, peripherals and accessories; and new technologies and techniques for input, output and interaction. He received a PhD in computer vision and robotics from the University of Cambridge and is a member of the IEEE and ACM. Contact him at shodges@microsoft.com.

Stuart Taylor is a Research Software Development Engineer at Microsoft Research, Cambridge. He has extensive experience of developing new hardware and software technologies in an industrial research environment. He received an MSc in Computing Science from the University of London. Contact him at stuart@microsoft.com.

Nicolas Villar is a Researcher at Microsoft Research, Cambridge where he focuses on the development of new hardware platforms and tools to enable technical innovation. He received a PhD from the University of Lancaster in ubiquitous computing. Contact him at nvillar@microsoft.com.

James Scott is a Researcher in the Sensors and Devices group at Microsoft Research Cambridge, UK. His research interests span a wide range of topics in ubiquitous and pervasive computing, and include novel sensors and devices, mobile interaction, rapid prototyping, wireless and mobile networking, energy management, and security and privacy. He received his PhD from the University of Cambridge in 2002, and is a senior member of ACM and a member of IEEE. Contact him at james.scott@microsoft.com.

Dominik Bial is a research assistant and PhD student at Paluno, the Ruhr Institute for Software Technology at the University of Duisburg-Essen in Germany where he also obtained an MSc in Software Systems Engineering. His research interests are future appliances and systems, software engineering and the future Internet, especially the Internet of Things. He can be reached at dominik.bial@stud.uni-due.de.

Patrick Tobias Fischer is a PhD student in HCI at the University of Strathclyde, Glasgow, UK. His research focuses on situated public interfaces in urban environments that have performative aspects. He has an MSc in novel input technologies from the Cologne University of Applied Sciences. He can be contacted at fischer@cis.strath.ac.uk.

Mailing addresses for complimentary copies

Steve Hodges, Stuart Taylor, James Scott, Nicolas Villar: Microsoft Research, 7 JJ Thomson Ave, Cambridge, CB3 0FB, UK.

Dominik Bial: Paluno - The Ruhr Institute for Software Technology, University of Duisburg-Essen, Gerlingstraße 16, 45127 Essen, Germany.

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE

Patrick Tobias Fischer: Strathclyde University, 26 Richmond Street, Livingstone Tower Lv.11.01, Glasgow, G1 1XH, UK.

Digital Object Indentifier 10.1109/MC.2012.394 0018-9162/$26.00 2012 IEEE