An Architecture for Web-Enabled Devices

J.J. Lukkien*, M.F.A. Manders, P.J.F. Peters and L.M.G. Feijs
Eindhoven Embedded Systems Institute
Eindhoven University of Technology

Abstract *Connecting other devices than workstations to the Internet is becoming commonplace now. We give a taxonomy of such devices based on the integration of this connection with the functionality of the device. Then we propose a generic architecture for Web-connected devices based on Internet standards.*

Keywords: embedded system, architecture, interface, protocol

1 Introduction

In this paper we study the architecture of Web-connected devices. In the most general view these are devices other than desktop computers that are connected to the Internet. Our focus will be on what are called traditionally: *embedded systems*, i.e., systems containing computer hardware and software to control their functionality. Examples are home appliances (radio and TV sets, washers, ovens etc.), computer hardware components like a printer a mouse or a screen, and telecommunication equipment like mobile phones. We call the embedded hardware and software of such a system the *Embedded Computer System* (ECS).

The paper is organized as follows. First we give a taxonomy of network enabled devices. This gives some insight in the current status and some perspective. Then we look at what Internet adds to this and why. The fourth and fifth section deal with the architecture. In section six we discuss the used protocol. We end with some conclusions. As a running example we use a garden light that can be monitored and controlled through the Internet (see Figure 1).

*Contact the first author for details. Full address: Eindhoven Embedded Systems Institute Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, Tel. +31 40 2478207, Fax. +31 40 2451733, Email: j.j.lukkien@tue.nl

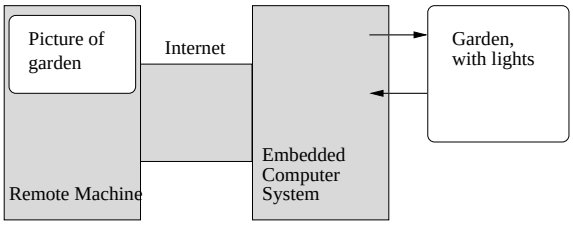

Figure 1: *Our running example is the simple system in which a set of garden lights are controlled across the Internet.*

2 A taxonomy of network enabledness

A network connection is an important addition for an embedded system. Without it there is no contact between the outside world and the ECS other than through the user interface of the device, if any. The functionality is determined by the manufacturer and one could simply regard the ECS as an implementation aspect which is hidden from the user (and rightfully so).

Using network connections is an evolutionary process and both the devices as well as their designs go through several stages. We have made a taxonomy which is derived from similar taxonomies that are sometimes used for the penetration of Internet. A taxonomy is useful in that it both gives an understanding of the current status and a perspective for improvements and extended functionality. In addition, it provides a context for comparison and is therefore a useful tool for a designer. A well-accepted taxonomy is the CMM model of the software construction process [1, 2]. An example of taxonomies for algorithms is given in [3]. For network connected devices we discern the following five stages.

Network unaware The device is monolithic, the ECS is unreachable and invisible.

Network aware It is found to be advantageous to retrieve certain information from the device. The most obvious type of information is *status information* and the most important ap-

plication is finding erroneous behavior as well as preventing such behavior. An example is a machine in a factory that records how often it is used and the wear and tear of its parts. Starting point is the fact that the device already contains many technical features, viz, the ECS. These features often determine the final design of the network connection. The used protocols are typically proprietary using already existing connections. The device is not necessarily connected to a network; it can also be inspected by a dedicated computer. E.g., the machine in the above example can store information somewhere internally where it can be retrieved through a serial interface .

Network connected The possibilities increase significantly when the cycle *embedded system → user → embedded system* is closed. To this end the device must be on-line while it is being used, i.e., it must be connected directly to the network. This extends the possible use to *interaction* with the device. Among others, this implies that it becomes possible to retrieve designated information rather than an off-line search in a large information dump. In addition, some control is possible.

The modifications within the embedded system are limited. The most important addition is a command interpreter supporting "remote monitor & control" functionality. This interpreter could run on an extra processor that is added for this purpose and translates the commands to the protocol of the embedded system, i.e., it serves as a protocol gateway to map the proprietary protocols to more standardized ones. Usually, dedicated lines are used for communicating with this type of system (e.g., telephone lines). We don't have any integration just yet: the network connection is an addition afterwards. The embedded system does not "know" about it. Our garden lights of Figure 1 are in this category.

These first three categories are illustrated in Figure 2.

Network centered When a network connection is part of the device right from the start, its design changes. With respect to the hardware this means that the "network part" is integrated with the ECS. This may lead to a new choice of hardware. This choice is further influenced by the requirements of the software that implements the functionality we sketched under "network connected". New aspects are that the network connection will become even more standardized (fewer proprietary protocols, following market trends) and new options are added like, for example, the possibility to reload the software of the ECS or to debug it. In addition, some intelligence can be added like reporting failures, giving a warning for wear-out or for aggregating any information that is leaving the system. The principal difference with the previous stage is that no longer the non-networked system is taken as a starting point hence no gateway is needed. Examples of systems within this class are organizers and last generation mobile phones.

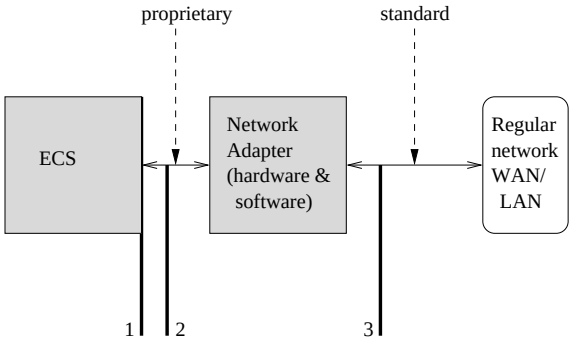

Figure 2: *Several views on the network connection of the ECS depending on where system boundaries are placed. In case 1 the system is network unaware; in case 2 it is network aware and in case 3 it is network connected. See the discussion in the text.*

Fully networked In this final stage not only the design but also the *functionality* of the product is determined by the fact that it is network connected. The embedded system uses the network to improve its own performance. In particular, it communicates with other *systems* rather than only with a single human user. For example, weather information available on the network can be used by the heating system of a green house. Functionality of individual system components can be made available on the network. E.g., a digital camera may find a screen for displaying or a place to unload stored pictures; an organizer may find a printer, etc.

A key notion here is the idea of *embedded intelligence*. It is in this area that the most futuristic images are sketched and it is mainly explored by describing scenario's. Also, here are the most challenging questions with respect to advantages and disadvantages of this technology (see e.g., [4]).

We do not expect that this final stage is useful for

all devices. It depends, in general on the stand-alone functionality of the device: the more general the service it provides, the more networked it becomes. For example, although we expect something as simple as a power outlet to be controllable through a network we do not expect it to be a network terminal with stand-alone behavior.

When devices become more networked it is important that their configuration is as easy as possible. For network enabled devices we still usually have that there is a single user. However, fully networked devices communicate with potentially many users and systems. In addition, it is expected that there will be very many of them and they will be switched on and off and moving around. This rules out the possibility of manual configuration and, therefore, so-called *zero configuration* ([5, 6]) is vital.

3 The extras of Internet

The evolution of "networked devices" has a technological background and is influenced strongly by financial and economic issues (e.g., who earns money?, who pays the bills?). In particular, the feasibility of fully networked devices relies on an existing wide-area network and sophisticated standards. Without these, a manufacturer would need to do the entire job himself which is a too large investment. Internet and its standards are a step in the right direction. It has become possible now to develop networked devices which connect to just about every computer. The Internet Protocol is rapidly becoming the widest accepted standard for information transportation and free software for this purpose is becoming available now. In short, Internet supports the required *standardization* and *interoperability*.

The technological developments can be summarized in three "laws": those of Moore, Metcalfe and Gilder [7, 8, 9, 10]. Moore's law states that the number of transistors per square millimeter chip area doubles every 18 months. The result is that computer hardware becomes virtually costless, which explains the enormous growth in embedded systems. Metcalfe's law refers to the use of connecting systems in a network: the value of a network grows as the square of the number of nodes, i.e., it grows as the number of node pairs connected by the network. Gilder's law is a statement about the expected growth of Internet: Internet bandwidth triples every 18 months. It does not have so much a technical background as the other two but it forces one to think about a situation in which bandwidth is no limitation.

The current situation is that the evolution of networked devices is not limited so much by technology but by other factors. Home appliances, for example, are replaced on a 7 to 12 year timescale. Connecting devices to the Internet is not trivial yet as there is no well-established standard for home networking (the "last meter" problem). Current developments address only part of the spectrum in isolation (e.g., domotica solutions or HAVi). Finally, questions regarding security and privacy are to a large extent unanswered.

4 An architecture for Web connectivity

In the project "Web-enabled Embedded Systems" at EESI we look in particular at Web-connected devices (the third stage in the taxonomy). The principal applications are *remote monitoring and control* which amounts to viewing and controlling the embedded system from a distance. Our interest is mainly in the architectural and design-related issues and we want to have a more general approach than just implementation on an ad-hoc basis. Therefore, we have isolated a number of trade-offs from a series of example implementations.

Generality We want to be able to re-use our design for fairly different embedded systems. At a high level of abstraction we want to re-use the system architecture but we would also like to re-use parts of the hardware and the software.

Principle of correspondence
A user should have a similar experience when using the embedded system across the Internet as when he uses it directly. The main advantages of this are ease of use and better acceptance: why learn something new? On the other hand, the remote connection gives us also the opportunity to change the user interface, for example, to improve it (think about programming a video recorder) or to use facilities not available in the original system interface (e.g., history information).

This principle can be used in the design as well: whenever a model of the device has to be build in software, relevant details are modeled as well. We come back to this later.

Platform independence The remote user can approach the embedded system from any platform and we do not want to impose too many limitations beforehand. If we do not want to maintain many different versions of the software we should use accepted Internet technol-

ogy for platform independence. Good candidates are an HTML browser and/or the language Java (which are available for just about any platform).

How to connect? The embedded system has an Internet address which can be used to connect to it. If we use a browser we can reach it through a standard URL. At the downside here is that the connection goes via HTTP, the HyperText Transfer Protocol which has a limited expressive power.

Performance and workload The ECS has limited resources. Therefore, we want to limit the amount of work that needs to be performed and the amount of memory that is required. In particular, we want to avoid to run a graphical user interface from within the ECS. More generally, we want to perform demanding computations at the user side as much as possible. Therefore we require a more flexible software module than a browser. An applet running within the browser can do this. An applet running within the client's browser is preferred over a client-side application, because of the platform independency of the applet, while both are able to provide the desired level of freedom. Notice that this also introduces some disadvantages: information protection, which can be used as part of HTTP is no longer available directly and therefore, must be taken into account explicitly.

Choice of protocol The HTTP protocol has limited expressive power. It is based on a request-reply scheme and the opportunities for the server to push unrequested information to the client are limited and ill-supported. This is an illustration of a more general principle: it is not a good idea to stretch the use of a protocol beyond the purpose what is was designed for. It is then better to go down the protocol hierarchy and to use one of the basic transport protocols: UDP or TCP. Here we deviate from a fairly common approach to try to realize everything with just an embedded Web server. Since plain HTTP does not allow us to choose between TCP and UDP, this choice also calls for a more flexible software module than a browser.

The resulting system architecture is given in Figure 3. After initialization, device monitoring and control takes place through the cooperation of an applet and the device server which is one of the architecures described in [11]. An important benefit of this architecture is the freedom to position

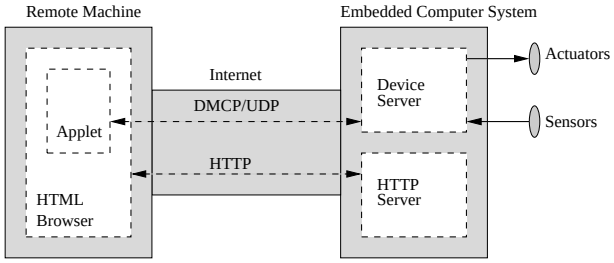

Figure 3: *A simplified overview of the combined hardware and software architecture. The closed lines and filled rectangles denote the hardware of the system, viz., two computer systems, Internet, sensors, actuators and wires connecting them. The software components and logical connections are denoted by dashed lines and white rectangles. They include a Web browser at the user side and a Web server at the ECS for setting up a connection. The actual monitoring and control is done using a device server at the ECS and an applet within the browser. They communicate using the Device Monitor and Control Protocol (DMCP) which runs on top of UDP.*

tasks wherever that is convenient. For example tasks, which demand much processing-power may now be placed at the client-side, which has typically less resource-constraints. This allows us to use less powerful, i.e. cheaper, processors in our embedded system. The applet can be replaced by a client-side application, as we mentioned before. Besides the platform independency, there is also a security benefit. Applets are restricted in their network access and can only connect to their originating host. Furthermore applets run in a sandbox, which means that they cannot perform system calls. The applet's tasks are as follows.

1. Initialize the connection with the device server.
2. Provide the user with the appropriate user-interface.
3. Translate the users input into DMCP commands and send them to the device server.
4. Interpret and translate incoming DMCP commands and present them to the user.

The device server can be written in any language, ranging from a servlet started by the embedded server to a small program written in assembly. In the latter case the embedded server can be very small as its only task is sending the applet. The tasks of the device server are threefold.

1. Make sure that the device remains in a safe (i.e., non-hazardous) state

2. Translate DMCP commands into actuator control and translate sensor information into DMCP commands.

3. Perform *session control* where a session is defined as a series of DMCP commands that should be executed without interruption. This includes authorization aspects.

Notice that due to the Internet connection the device becomes a multi-user device. The device server must be able to deal with this.

5 Generality and the principle of correspondence

When we take the principle of generality to the extreme we would generate a specialized version of Figure 3 based on a description of an embedded system. This is one of the project goals but we assume that it can be reached only for applications within a limited domain. This means that we also look at the prospect of re-use for the individual system components, which are the interface to the outside world (sensors/actuators), the applet, the device server and the protocol across the Internet. It appears that the choices at the interface (the first two components) determine the design of the last two components to a large extent. A possible solution for the interface to sensors and actuators which is generally applicable is to serialize all actuator and sensor information into bitstreams. This solution is also common practice in hardware (serialization leads to small data busses). An example along these lines is found in [12].

As explained before the principle of correspondence leads to an implementation in which the remote user has the same look-and-feel of the embedded system as when he is using it directly. This principle has a counterpart in the applet: the physical objects that can be manipulated through the actuators of the system and whose state can be monitored through the sensors are modeled as objects in the applet. These objects have a local state which is composed of an *actuator part* which is controlled by the user through the interface of the browser and a *sensor part* which is controlled by the embedded system through DMCP messages. For the applet, there is a reason to take action whenever these two are unequal.

With respect to user interaction, it is important that the user gets relevant feedback about what he does and how the ECS has responded. The user interface should make this distinction clear. For example, when a user presses a button in order to switch on a light he must be able to notice that he pressed a button (the actuator state). The light representation in the user interface must switch on only as a result of changes in the sensor state.

Class	Type	Parameters	Reply
Session	hello	user	ack
	bye	user	ack
Control	write	user, actuator set, value set	ack
Monitor	read	user, sensor set	ack / info
	subscribe	user, sensor set	ack
	unsubscribe	user, sensor set	ack
Response	ack	message type, error status	
	info	sensor set, value set	

Figure 4: *Overview of DMCP.*

6 DMCP

The Device Monitor and Control Protocol was devised for the communication between the applet and the device server (see Figure 3). On the one hand one can argue that this protocol is completely internal for the system and, hence, we can choose it arbitrarily depending on the embedded system at hand. On the other hand we want to use this protocol in the future for communication *between* embedded systems as we move up in the hierarchy sketched in section 2. That is why we want to make it as general as possible. The work in this area is currently under development.

The exchanged messages consist of a type and optional parameters (see Figure 4). The user id sent with all messages is present to enable message replies. One could argue that since probably IP will be the transport protocol, a user id is superfluous since IP provides that already. In principle this is right, but we don't want to be limited to using the IP protocol only. In addition, the user id may contain more information than just the address or may differ from this IP address. Following current Internet practice the protocol is verbose. Each message sent to the device server is answered by an *ack* message containing the message type and, possibly, an error status. An error-free *read* message is answered by and *info* message containing the relevant information. Messages fall into four classes.

Session Setup and control the connection between

a user and the embedded system. The session concept is the way we deal with limiting user access to an ECS. In most cases, control of an ECS will be in the hands of a limited number of users (in most cases only one), so access to the ECS has to be limited in some way. This is done using the session control messages. Users that are not given access will receive an *ack* with an error message after sending a *hello*. The *bye* message informs the ECS of the end of a session. Sessions are not always needed: it highly depends on the ECS. Consider e.g. a temperature sensor that only allows reading the temperature. It could have numerous users reading its status. In general the session concept will be needed

- if multiple users can interfere thereby invalidating the view of the ECS state of one of the users, or
- if access to the ECS is limited because of security reasons.

Control Change the value of the actuators. There is only a single message type, the *write* message. It has some way of addressing the actuator that is controlled as well as the value it must receive. This is considered system specific and is therefore left unspecified. Control messages can be given only within a session (if relevant).

Monitor Inquire about the current status of the system. A direct inquiry is through the *read* message which obtains sensor information in a format similar to the *write* message. In addition we have the opportunity to subscribe to changes in sensor status. Monitor messages may be given inside or outside a session although the system may impose restrictions. For example, the restriction may be imposed that subscription messages can be done only during a session.

Response Response messages can be sent only by the ECS. We have a two message types, the *info* message transferring state information and the *ack* message to acknowledge messages and to inform about an error. The *info* message are generated as response to *read* messages sent to the system or as a result of a sensor change to the subscribers of that sensor.

A more abstract way to look at the notions of "sensor" and "actuator" in the protocol description is to regard them as state variables that can be read or written. For example, if the possibility of reading the current value of the actuators is also required these should be added as "sensors" as well.

The protocol uses a UDP connection. It could in principal run over TCP as well. The advantage of a TCP connection is mainly the guaranteed delivery of messages. The disadvantage is that an open point-to-point connection for each communicating partner must be maintained within the ECS. This amounts, in fact, to another notion of a session and in the protocol we would need to add connect/disconnect messages. UDP does not have this disadvantage but here messages might be lost. This is not solved in the protocol but left to the components that use it, e.g., by retransmitting after a certain time-out. Although the protocol is not stateless this is harmless, in principle, since the control messages do not refer to a previous state. The use of time-outs is necessary anyway because of the notion of a session. The ECS must be able to see whether a session is still "alive". In Figures 5 through 7 we have shown diagrams in the Message Sequence Charts format ([13]) depicting scenarios of message exchanges.

The actual implementation of the protocol may vary depending on the system. E.g. if the actuators of the embedded system are few and simple, the *write* message may be simplified by creating multiple specific messages for writing specific values to each actuator. The same reasoning is valid for other messages. If there are many actuators, or if actuators can have many values, it is better to add the actuator identification and value as a parameter to the messages to prevent message explosion.

7 Conclusion

In this paper we have introduced a taxonomy of Web-connected devices and we have developed a generic architecture for Web connectivity. We have discussed the design trade-offs which give a better understanding why this architecture is chosen. A new protocol for communication with such a device has been developed. Future research focusses on letting devices control each other through this protocol. We also want to relate it to new developments in this area, in particular, to Miscrosofts Universal Plug and Play standard [14, 6, 5].

References

[1] www.sei.cmu.edu/cmm/cmm.html.

[2] P. Jalote. *CMM in practice: processes for executing software projects at infosys.* Addison-Wesley, 1999.

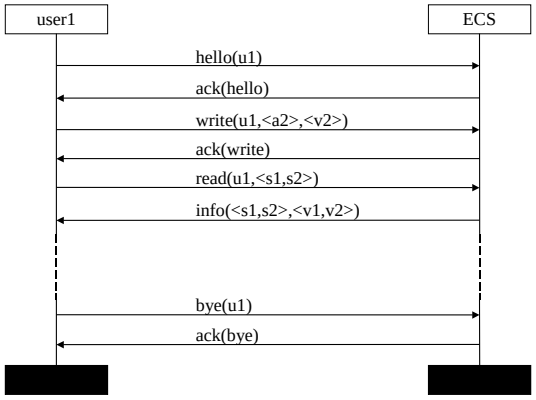

Figure 5: *MCS diagram of single user to ECS communication. User1 opens a session by sending the* hello *message. It receives an acknowledge containing just* hello *and no error. The* write *message that follows assigns* $v2$ *to actuator* $a2$*; it is confirmed by the ECS. Then a* read *message is sent inquiring information about the status of sensors* $s1$ *and* $s2$*. The* info *message sent back contains the requested information. The session is closed by the* bye *message which is confirmed as well.*

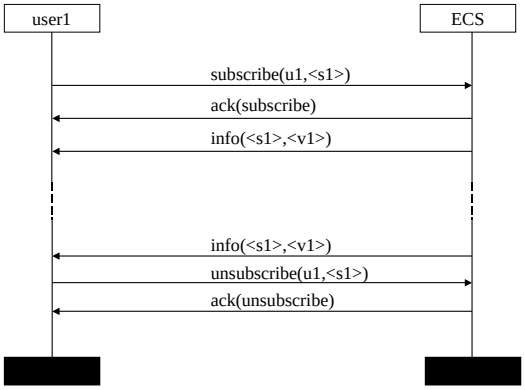

Figure 6: *MSC of single user subscribing to sensor changes. The* subscribe *message informs the ECS that user1 wants to receive* info *messages upon changes of sensor* $s1$*. The subscription is indeed performed as is indicated by the positive acknowledgement. Then the user closes the session. At some point in time user1 receives an* info *message indicating a change in sensor* $s1$*. After that user1 sends an* unsubscribe *message to inform the ECS it no longer needs to send messages sensor s1 changes; this is confirmed.*

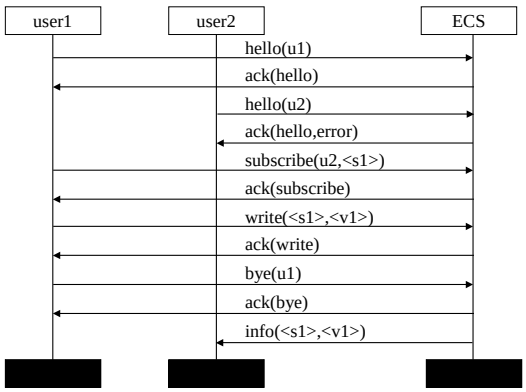

Figure 7: *MSC of two users sharing access to the ECS. Because one user blocks access of the other one we now have several failures in the communciation. Notice that one user may subscribe another one.*

[3] B.W. Watson. *Taxonomies and toolkits for regular language algorithms*. PhD thesis, Eindhoven University of Technology, Einhoven, the Netherlands, September 1995.

[4] Scientific american, October 2000.

[5] E. Guttman. Zero configuration networking. In *Proceedings of the INET 2000 conference*, July 2000. www.isoc.org/inet2000/-cdproceedings/3c/3c_3.htm.

[6] www.ietf.org/html.charters/zeroconf-charter.html.

[7] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, April 1965.

[8] www.iso.gmu.edu/ rschalle/moorelaw.html.

[9] Metcalfe's law and legacy. www.seas.-upenn.edu:8080/ gajl/metgg.html.

[10] G. Gilder. Telecosm: how infinite bandwidth will revolutionize our world, 2000.

[11] M. Barr. Developing embedded software in java. In *Proceedings of the Embedded Systems Conference, Part 1*, 1999.

[12] B. Kainka. A universal measurement interface (in dutch). (434):X2–X6, December 1999.

[13] M.A. Reniers. *Message Sequence Charts: Syntax and Semantics*. PhD thesis, Eindhoven University of Technology, 1998.

[14] www.upnp.org.