Minimalist Architecture to Generate Embedded System Web User Interfaces

Fernando Pereira[1,2,3] and Luís Gomes[1,3]

[1] Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia – Portugal
[2] ISEL, Instituto Superior de Engenharia de Lisboa, – Portugal
[3] UNINOVA - CTS – Portugal
fjp@deea.isel.ipl.pt, lugo@fct.unl.pt

Abstract. This paper presents a new architecture to semi-automatically generate Web user interfaces for Embedded Systems designed using IOPT Petri Net models. The user interfaces can be used to remotely control, monitor and debug embedded systems using a standard Web Browser. The proposed architecture takes advantage of the distributed nature of the Internet to store all static user interface data and software on third-party Web services (the Cloud), and execute the user-interface code on the user's Web Browser. A simplified protocol is proposed to enable remote control, status-monitoring, debugging and step-by-step execution, minimizing resource consumption on the physical embedded devices, including processing load, memory and communication bandwidth. As the user interface data and code are kept on third-party Web services, these resources can be shared among multiple embedded device units, and the hardware requirements to implement the devices can be simplified, leading to reduced cost solutions. To prevent down-time due to network problems or server failures, a fault-tolerant topology is suggested. The distributed architecture is transparent to end-users, observing just a Web interface for an embedded device on the other side of an Internet URL.

Keywords: Embedded-systems, Petri-Nets, Web User Interface.

1 Introduction

The widespread dissemination of the Internet and the mass production of telecommunication technology has significantly reduced the cost barrier to add Internet connectivity to the most inexpensive embedded-system devices, ranging from home appliances, medical and health monitoring devices, surveillance and security equipment, in-vehicle systems, to industrial machinery. However, the traditional Internet connectivity implementation strategies generally lead to increased product complexity, longer development time and increased hardware requirements, including more memory to store images and multimedia files and higher processing power to execute user interface code, with the corresponding implications on power-consumption and battery life.

L.M. Camarinha-Matos, S. Tomic, and P. Graça (Eds.): DoCEIS 2013, IFIP AICT 394, pp. 239–249, 2013.

This paper proposes a new architecture to overcome these limitations, taking advantage of the distributed nature of the Internet and the processing capabilities offered by modern Web browsers. All static data files are stored on external Web servers and the user-interface code is executed directly on the user's personal computer Web browser, greatly reducing the embedded devices hardware requirements and contributing to minimize bandwidth consumption, as many user interactions are dealt directly by the user interface code running on the browser.

To implement the new architecture, a new protocol is proposed to establish the communication between the user-interface code running on the browser and remote embedded devices, taking advantage of AJAX (Asynchronous Javascript and XML) principles. The same protocol used to implement the Web user interfaces can also be used to enable remote monitoring, debug and step-by-step execution of embedded system controllers running on physical devices, or simulated on personal computers.

The proposed architecture is applied to embedded systems designed using the IOPT Petri-net modeling framework [1] and the associated Web-based IOPT-tools [2] tool-chain (http://gres.uninova.pt), including an editor to design IOPT models, a model-checking framework based on a state-space generator and a query system, and automatic "C" and VHDL code generators to produce the controller implementations.

The user interfaces are designed using an Animator tool [3] that permits the interactive rule-based definition of Animated Graphical User interfaces for embedded-systems. This tool has been previously used to design and generate simulation control-panels and graphical user interfaces for systems running on FPGA (Field Programmable Gate Array) reconfigurable devices. Using the Animator tool, the user-interface designer creates a set of screen background images, used to implement application dialogs, and sets of static or animated icons and images. Each dialog contains a table of rules associating internal state variables (IOPT Marking) and Input/Output Signals with the appearance and screen location of the selected icons and images, creating an animated SCADA-like user interface that reflects the system-state in real-time. To control the system from the Web interface, Input Signals can be associated with icons and images activated using mouse clicks to implement bidirectional user-feedback.

The new architecture proposes the execution of Animator-generated user interfaces inside a Web browser, converting the Animator rules into equivalent Javascript code and establishing the communication with the physical systems using remote procedure calls over a protocol based on AJAX *XmlHttpRequests*.

The creation of Debug and monitoring interfaces has been presented in [4], with the automatic generation of Animator screens that directly depict the corresponding IOPT models and the automatic generation of rules that display the system status in real-time, including place marking, transition firing readiness and the state of Input and Output Signals. However, in order to fully support remote debug sessions over the Internet, the communication protocol will be extended with additional remote procedure calls implementing step-by-step execution, continuous execution, system reset, force input signal values and the definition of breakpoints associated with Transition firings.

Finally, this paper proposes a new type of Transition, called the Test-transition, used during the debug and simulation development phases. Test-transitions differ from standard IOPT Transitions because they are not allowed to change system behavior in any form. This way, the new transitions can be freely added to existing models without the risk of accidentally introducing behavioral modifications, to define breakpoints associated with new conditions that were not verified in the original models.

2 Related Work and Research Innovation

Over the past decade, Internet enabled embedded devices with Web interfaces have been offered by commercial systems and were implemented on many research prototypes. The traditional architectures used to implement these solutions have resorted to full-featured Web servers running over embedded operation systems, as embedded Linux [5] and QNX [6], using standard interface technologies like common-gateway-interface (CGI) to control the physical embedded devices.

However, solutions based on complex operating systems require advanced microprocessors and occupy large amounts of memory, including many megabytes of RAM (Random Access Memory) and mass storage devices to store operating system files. The Web interfaces are generally created using standard Web page authoring tools and the connection to the physical embedded systems are manually programmed. All files used by the Web interface, including images and scripts are usually stored in the device. Due to these requirements, Internet connectivity has usually been skipped from the least expensive devices.

The new solution presented in this paper has many advantages over traditional technologies. To start, the Web user interfaces are semi-automatically generated using the rule-based Animator tool [3] and the Debug interfaces are fully automatically generated [4] without the need to manually write any code. As the static files are stored in external Web servers and the user interface code (Animator rules) is executed by the user's Web browser, the computational requirements of the embedded controller are largely reduced and the need to employ complex operating systems is avoided, allowing the addition of internet connectivity and Web interfaces to the most inexpensive devices without a significant cost increase. Instead of requiring 32-bit microprocessors, the proposed minimalist architecture can be implemented with simpler 8-bit embedded micro-processors using small TCP/IP stacks as uIP and LwIP [7] that require just tens of Kilobytes of RAM and can entirely fit inside the memory blocks offered by FPGA devices without external RAM. The usage of 8-bit micro-processors also contributes to reduce FPGA resource consumption, enabling the choice of smaller and less expensive reconfigurable devices.

In addition to the automatic generation of Web user interfaces, the proposed communication protocol also permits remote debugging and step-by-step execution of embedded-system controllers running on real hardware devices, allowing long distance troubleshoot and maintenance operations over the internet. As the graphical debug interfaces automatically generated by the PNML2Anim4Dbg [4] tool can be directly presented in the new architecture, it is possible to monitor the state of remote embedded-systems in real time, observing the graphical evolution of the underlying IOPT Petri net model. Contrary to traditional remote administration tools, this solution offers a high degree of intuitiveness and user friendliness, as the system designer operates directly on the Petri net model used to design the original system.

Other Petri net tools, including CPN tools [8], CPN-Ami [9], Renew [10] and others have implemented debug and step-by-step execution tools, but as these classes of Petri nets are autonomous, the scope of these tools is generally restricted to simulations running on personal computers and not for final implementations. Inside these simulation tools, some authors have also defined the concept of breakpoints associated with transitions and changes on place-markings [11], but the new concept of Test-transition presented in this paper offers many advantages over these solutions as it enables the definition of generic breakpoint conditions.

Finally, the new architecture was built on top of previous work, starting with the definition of the IOPT Petri Net class [1], IOPT design tools [2], automatic code generators and the Animator Tool [3]. The proposed architecture enables the porting of previous Animator-designed user interfaces that were executed inside simulations or on FPGA hardware [12] [13], to a distributed Web/Internet platform.

3 IOPT Petri Nets

The characteristics of the IOPT Petri net class [1] were selected to support the design of embedded-system controllers. Beyond the Places and Transitions inherited from classical Petri nets [14], IOPT nets also contain a set of non-autonomous properties used to specify the interface between the controllers and the external world. This interface comprehends Input and Output Signals, that can hold Boolean logic values or Integer range values, and Input and Output Events associated with changes in Signal values. Figure 1 displays an example IOPT Model implementing a UART transmitter hardware module, edited with the IOPT-Tools model editor.

The model presented in Fig 1, has three Places (yellow circles), four Transitions (cyan rectangles), three Input Signals (cyan circles), five Output Signals (green circles), and one Output Event (green triangle). Associated with Places there are Output Expressions that assign values to Output Signals whenever these Places are marked. Guard conditions, associated with Transitions, inhibit the firing according to the value of Input/Output Signals, Literals and Place marking. Transition firing is also triggered by Input Events, and can produce Output Events that perform changes in Output Signals. For example, transition ***TCount*** raises a ***Cntr*** Output Event, used to count the number of bits transmitted by incrementing the value of the ***Cnt*** Output.

In order to ensure determinist execution, IOPT nets employ maximal-step execution semantics, where every transition ready to fire will immediately fire on the next execution step. Firing conflicts, when more than one transition is simultaneously ready to fire, but the number of available tokens is not enough to fire all of them, can be solved by assigning different priorities to each conflicting transition. A state-space generation tool offered by IOPT-Tools can be used to automatically detect conflicts, deadlocks and to calculate the maximal and minimal bound of each Place.

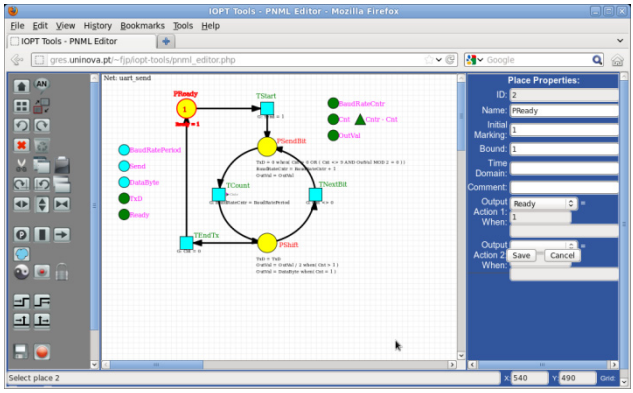

Fig. 1. IOPT-Tools Editor (UART Transmitter IOPT Model)

4 Distributed Web User-Interface Architecture

Figure 2 displays the distributed topology of the proposed embedded-system's Web user interface architecture, employing the Web browser running on the user's personal computer, an external Web server and the embedded-system.

The user interfaces are designed with the Animator tool [3], producing a set of screen dialogs, image files and animation rules. Image files hold the background pictures of each screen and graphical representations of icons, buttons and animation frames. Animation rules define when images are show or hidden according to the instantaneous embedded-system state, including the Place marking, Input signals and Output signals.

As the Animator tool stores the animation rules under XML files, these rules can be easily translated into equivalent Javascript code using XSL (Extensible Style Sheet Transformation) transformations or other XML processing technologies. The resulting Javascript code is executed inside the Web Browser, removing the computational load of the user interface code from the physical embedded device.

All static files, including images and Javascript code, are stored in external servers, eliminating the need to store large files on the embedded device. By moving the user interface code and file storage to external computers, the embedded-system controller can be implemented using minimal hardware specifications. The embedded system controller only needs to implement a micro HTTP server, used to answer remote procedure call from the Web browser and transmit information about the current system state (almost) in real time.

5 Internet-Enabled Embedded-System Internal Architecture

The main goal of the architecture proposed in this paper is the ability to add Internet support to embedded-system devices with minimalist hardware specifications. Figure 3 presents a possible hardware architecture that fulfills these goals.

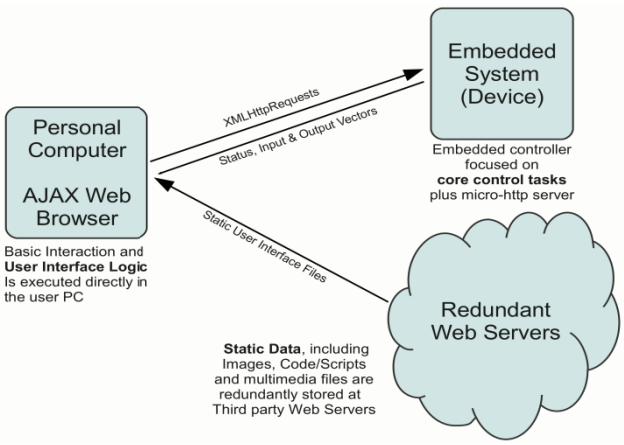

Fig. 2. Distributed Web GUI Topology

The proposed hardware architecture requires just a minimal set of components to implement the TCP/IP communication protocol over Ethernet networks. The entire system can be implemented in a single FPGA (or ASIC), with the addition of an external Ethernet Physical Layer Integrated Circuit (PHY) and the respective magnetics and connector, or equivalent components to support wireless networking.

Resource consumption inside the FPGA/ASIC device to implement TCP/IP over Ethernet requires a standard Ethernet MII/MAC (Media Independent Interface / Media Access Controller) module and an embedded microprocessor, used to execute the TCP/IP stack and a mini Web-server. There are many microprocessor cores offered by FPGA vendors and many open-source processor cores publicly available. As the open-source LwIP (Light Weight Internet Protocol) is coded using the "C" Programming Language, the only processor choice requirements are the availability of a "C" compiler and a 16 bit memory address space (or larger), in order to store data packets, plus the 40Kb LwIP code and a small HTTP command interpreter. These requirements cover a wide range of microprocessors, ranging from 8-bit Zilog Z80 clones to more advanced 32 bit MicroBlaze cores.

A clock management unit is also employed to allow debugging and step-by-step execution of the embedded-system controllers, by disabling the controller clock signal or sending individual clock pulses to run single execution steps.

The embedded microprocessor I/O bus is connected to the embedded controller core and is able to read signals containing the instantaneous values of Input Signals, Output Signals and Place marking. It can also define the value of Signals associated with Graphical User Interface objects. For example, an Icon or a Button inside an Animator screen can be associated with an Input Signal in the embedded system controller and when the user manipulates that button, a HTTP request is sent to the microprocessor to update the corresponding Signal value.

A set of optional internal signals is used to support remote debugging and maintenance operations, including a breakpoint mask, a forced-input-signal mask, a reset signal and three signals to control the clock management unit: stop, run and step.

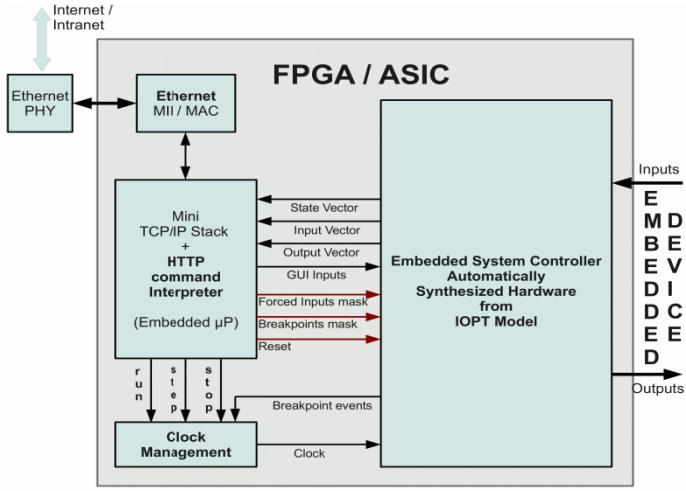

Fig. 3. Internal Controller Architecture

The breakpoint mask is used to enable breakpoints on selected IOPT Transitions. When a selected Transition fires, it will trigger a breakpoint event that stops the clock management unit. The forced-input-signal mask can be used to remotely force the value of Input signals read by the embedded controller, in order to bypass mechanical problems in the physical embedded devices or to easily force error situations and test the model behavior during these conditions.

6 Information flow over the Internet

Figure 4 presents the flow of information over the Internet between the Browser, the embedded-system and an external Web server. A typical interaction session starts when the user accesses the embedded-system URL (Web address) using a Web Browser. The embedded system answers with a Top/Start HTML page containing just a reference to a main script file stored in an external server. As soon as the main script is loaded and the user is successfully authenticated, it immediately starts downloading all Animator files required to execute the graphical user interface, including images and the scripts that execute animation rules. To obtain redundancy and prevent service failures in case the external Web server is not reachable, the Start HTML page might include multiple references to copies of the main script located in different servers that will load only if the first server does not respond during a predefined time interval.

After the start-up sequence is finished, a Web user interface main loop starts running, continuously requesting updated state information from the embedded-system, including Place marking and Input and Output Signal values. Using these values, the animation scripts will update the graphical user interface, almost in real-time, with an update rate configurable according to the available bandwidth and the requirements of each application.

As the static files stored in the external server are loaded in background, the distributed topology is transparent to end user that only sees the address (URL) of the embedded device.

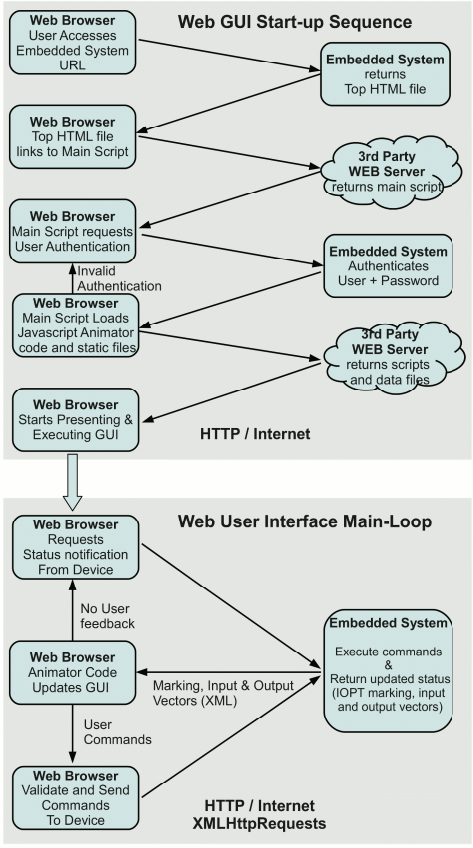

Fig. 4. Internet-IOPT Information Flow

7 The IOPT-Internet Protocol

The proposed Internet-IOPT protocol defines a set of commands, viewed as remote procedure calls over HTTP, used to establish the communication between the user interface code running on the Web Browser and the embedded-system HTTP command interpreter. It comprehends three types of procedures calls: 1) front-page and authentication; 2) status-monitoring and GUI interaction and 3) debug and step-by-step execution.

Start-page requests and user authentication are implemented with the standard HTTP commands used to serve static HTML files. User sessions can be implemented using a temporary HTTP cookie holding a random number that identifies each user after the authentication phase is successfully passed.

The second group of commands implements remote procedure calls that produce XML results, implementing the following methods:

getMarking() - read the instantaneous IOPT net marking vector
getInputs() - read the instantaneous input signal vector
getOutputs() - read the instantaneous output signal vector
getStateChanges(tm) - read «marking, inputs and outputs» changed since the previous read or wait «tm» seconds if there are no changes
setGUInput(sigName, value) - write the value of a GUI input signal.

All remote procedure calls are implemented using AJAX *xmlHttpRequests*, and the embedded-system's Web server answers to each command with XML files containing the requested values. The *getStateChanges()* method provides reduced bandwidth consumption, as it only returns values that suffered changes since the last read operation and will block when there are no changes during a specified timeout period.

Finally, the debug and step-by-step group of commands, used to directly control the clock management unit shown on figure 3, implement the following methods:

reset() - reset embedded-system controller state
stop() - stop execution
run() - continue execution
step(n=1) - execute «n» execution steps (by default n=1)
setBreakpoint(trId) - enable breakpoints on transition «trId»
clearBreakpoint(trId) - disable breakpoints on transition «trId»
forceInput(sigName, value) - force the value of a non GUI input signal
releaseInput(sigName) - release a forced input signal

8 Breakpoints and Test-Transitions

The proposed hardware architecture and communication protocol supports remote debug and step-by-step execution, including a hardware clock management unit that can stop or run continuously and is able to generate individual clock pulses to perform single execution steps. However, as complex embedded-systems often execute thousands of execution steps before reaching a critical situation being tested, step-by-step execution

may not be practical. Other embedded-systems simply cannot be run step-by-step because the controllers must respond to external input changes during very fast time intervals in order to prevent malfunctions and mechanical damages. For example, an automatic door controller must immediately turn off the door motor when the door is being closed and a presence sensor detects a person inside the door limits.

To solve this problem, the concept of breakpoints, usually employed in software debugging systems, was extended to the IOPT Petri net modeling framework, adding the possibility to assign breakpoints to Transitions. When a Transition fires, a Breakpoint event will be raised and execution is stopped before the Transition is actually fired. Observing figure 3, breakpoint events are connected to the clock management unit and will immediately stop the clock signal. This concept can also be easily ported to software implementations and may be applied on simulations or on embedded devices running on microprocessors.

Finally, the test conditions corresponding to the error situations being debugged may not directly correspond to the firing of any Transition existing in the model. As a consequence, it might be necessary to add additional new Transitions to the model, containing Arcs, Input events and Guard conditions that detect the (un)desired situations. However, adding new Transitions to an existing model will usually introduce changes to the behavior of the original model, potentially invalidating the debug conclusions, as the new model may behave differently from the original one.

To solve this problem, a new concept of Test-transition is proposed. A Test-transition has a set of restrictions that does not allow behavioral changes and can be safely used to define test/debug conditions associated with breakpoints. In order to achieve this effect, Test-transitions can only be connected to Test Arcs, cannot be connected to Normal Arcs and may not be associated with Output events. As a result, Test-transitions can only have input Test Arcs and cannot have output Arcs.

The concept of Test-transition has many advantages. First, Test-transitions can be safely added to any model in any configuration without the risk of introducing any behavioral changes. Second, Test-transitions can continue to be viewed as regular Transitions by all IOPT Tools, including the automatic software and hardware code generators, simulators and validation tools, without requiring additional development. Finally, Test-transitions can also be safely removed from models using automated filters, to generate final controller implementations without Debug code.

9 Conclusions and Future Work

The goal of this paper is to propose a new architecture to automatically add Web User Interfaces to embedded-system controller designed with IOPT Petri net models. This architecture employs a distributed topology to take advantage of the Web browser processing power and the storage capacity offered by external Web servers, in order to minimize the hardware requirements on the physical embedded-systems, implementing Web awareness without almost no additional cost.

This work is an extension to previous work, where the Animator [3] concept and tools were introduced. The proposed architecture can be used to automatically convert

existing user interfaces designed with the Animator tool, producing equivalent Web interfaces.

In addition to automatic Web interface generation, the same communication protocol was also extended to support remote debugging and step-by-step execution, offering the possibility to perform maintenance and diagnose problems over the Internet. The association of breakpoints to Transition firing and the new concept of Test-transitions also contribute to reduce test and debugging effort, simplifying the debug of systems where step-by-step execution would be impractical.

Although the proposed architecture was not yet implemented, all the components used in the architecture are currently disseminated technologies, including the suggested hardware platforms, the embedded TCP/IP protocol stack and the AJAX technologies used to execute the user interface code in the Browser. All the components employed are readily available, leading to the conclusion that the implementation of the proposed ideas is feasible. Future work may lead to the creation of prototypes based on the new architecture, with possible improvements to overcome technical difficulties that may occur during development.

Acknowledgments. This work was partially financed by Portuguese Agency "FCT – Fundação para a Ciência e Tecnologia", in the framework of project Pest-OE/EEI/UI0066/2011.

References

1. Gomes, L., Barros, J., Costa, A., Nunes, R.: The Input-Output Place-Transition Petri Net Class and Associated Tools. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN 2007), Vienna, Austria (July 2007)
2. Pereira, F., Moutinho, F., Gomes, L.: Model-checking framework for embedded systems controllers development using IOPT Petri nets. In: 2012 IEEE International Symposium on Industrial Electronics (ISIE), May 28-31, pp. 1399–1404 (2012), doi:10.1109/ISIE.2012.6237295
3. Gomes, L., Lourenco, J.: Rapid Prototyping of Graphical User Interfaces for Petri-Net-Based Controllers. IEEE Transactions on Industrial Electronics 57, 1806–1813 (2010)
4. Pereira, F., Gomes, L., Moutinho, F.: Automatic generation of run-time monitoring capabilities to Petri nets based Controllers with Graphical User Interfaces. In: Camarinha-Matos, L.M. (ed.) DoCEIS 2011. IFIP AICT, vol. 349, pp. 246–255. Springer, Heidelberg (2011)
5. de Souza, R.N., Muniz, D.N., da Silva Fidalgo, A.V.: Ethernet communication platform for synthesized devices in Xilinx FPGA. In: 2011 IEEE International Conference on Computer as a Tool (EUROCON), April 27-29, pp. 1–4 (2011), doi:10.1109/EUROCON.2011.5929377
6. QNX website, `http://www.qnx.com/developers/docs/6.3.0SP3/neutrino/user_guide/embedded_web_server.html` (accessed January 6, 2013)
7. Dunkels, A.: Full TCP/IP for 8 Bit Architectures. In: Proceedings of the First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys 2003), San Francisco (May 2003)

8. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Pratical Use. Basic Concepts, vol. 1. Springer, Berlin (1997)
9. Hamez, A., Hillah, L., Kordon, F., Linard, A., Paviot-Adet, E., Renault, X., Thierry-Mieg, Y.: New features in CPN-AMI 3: focusing on the analysis of complex distributed systems. In: Sixth International Conference on Application of Concurrency to System Design, ACSD 2006, June 28-30, pp. 273–275 (2006), doi:10.1109/ACSD.2006.15
10. Kummer, O., Wienberg, F., Duvigneau, M., Cabac, L.: Renew – User Guide, University of Hamburg, Department for Informatics, Theoretical Foundations Group, Release 2.2, (August 28, 2009)
11. Sadilek, D.A., Wachsmuth, G.: Prototyping Visual Interpreters and Debuggers for Domain-Specific Modelling Languages. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 63–78. Springer, Heidelberg (2008)
12. Moutinho, F., Gomes, L.: From models to controllers integrating graphical animation in FPGA through automatic code generation. In: IEEE International Symposium on Industrial Electronics (ISIE 2009), Seoul Olympic Parktel, Seoul, Korea, July 5-8 (2009)
13. Moutinho, F., Pereira, F., Gomes, L.: Automatic Generation of Graphical User Interfaces for VHDL based Controllers. In: ISIE 2011 – 20th IEEE International Symposium on Industrial Electronics, Gdansk, Poland, June 27-30, pp. 1491–1496 (2011), doi:10.1109/ISIE.2011.5984381, ISBN: 978-1-4244-9312-8
14. Reisig, W.: Petri nets: an introduction. Springer Verlag New York, New York (1985)