A Component-Based Technology for Hardware and Software Components

Luka Lednicki
Mälardalen Real-Time Research Centre
PO Box 883, SE-721 23, Västerås, Sweden
luka.lednicki@fer.hr

Ana Petričić, Mario Žagar
Faculty of electrical engineering and computing
Unska 3, 10 000 Zagreb, Croatia
{ana.petricic, mario.zagar}@fer.hr

Abstract

One of the challenges in development of embedded systems is to cope with hardware and software components simultaneously. Often is their integration cumbersome due to their incompatibilities, different specifications and different approaches in their development. In this paper we present a component-based technology for building distributed embedded systems consisting of both embedded hardware devices and software components. To obtain a uniform view on hardware and software we have developed a new component model – UComp. Our technology consists of UComp component model that allows treating remote devices as components, and a run-time framework that supports this component model when the system is deployed. To evaluate the principles we have developed a prototype tool that implements the technology and uses Universal Plug and Play (UPnP) standard for communication between system parts.

1. Introduction

With the continuous advancement of embedded computers their usage expands rapidly, and today a vast majority of computer nodes fall into embedded systems domain. Examples can be seen in environmental and industrial monitoring and control, home automation, and many other domains. These systems are built from various hardware units coupled with software components which are frequently distributed over several nodes. As an example we can take weather forecast systems consisting of numerous smart sensors, and communication and computational nodes dispersed in a large geographical area.

One of the ways to cope with this rising complexity of systems is by using the component-based approach for their development. While the general purpose component-based technologies, like COM [5] or .NET [3], provide solutions for high level applications (for example desktop or web applications), component technologies for embedded systems (e.g. SaveCCM [1] and Koala [7]) are mostly intended for development of software components embedded into hardware devices. A problem arises when trying to connect the two in complex systems consisting of both high level software, and low level embedded components that are closely connected to the hardware.Some standards like OPC [4] and AUTOSAR [8] enable a degree of cooperation between software and hardware components, However, we want to provide a component model and framework that would make it possible to handle hardware and software components during both design and run-time phases of a system. The focus of our work is on distributed systems whose functionality is implemented using various devices connected to a computer network. These devices may either be physical, i.e. realized using hardware, or virtual, i.e. realized using software applications.

Run-time modification would enable late deployment of new embedded devices, or replacement of existing ones. Also we want to eliminate the need for specialized device drivers by automatically generating components that conform to our component model from the device descriptions.

The purpose of this paper is to present our UComp technology which provides such an environment.

In Section 2 we present our component model while its realization and run-time framework is described in Section 3. Section 4 concludes the paper and states the possibilities for future work.

2. The UComp Component Model

To achieve our goals we have developed a new component model, UComp and its supporting framework.

Component interfaces. Interfaces of UComp components are defined by their input and output ports. Ports are used to exchange data and control (triggering) signals. Data and triggering signals from output port of one component can be directed to input ports of one or

more components. Graphical representation of a sample UComp component named *Component A*, together with input ports *a*, *b* and *c*, and an output port *out* is shown in Figure 1.

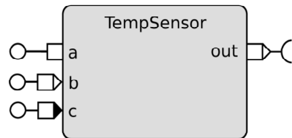

Figure 1. Graphical representation of an UComp component and its ports.

Ports of UComp components are defined by their names and data types. While output ports handle only one type of data, input ports can handle a number of different data types. UComp model itself does not specify or limit the type of the data.

Any change in the value of the data of an output port creates a trigger signal on that port. A port can also be configured to handle no data; in this case it is used for triggering purposes only.

Connections between Ports. Whenever a component sets new data to one of its output ports, the port propagates this data and triggering signals to all input ports connected to it. Data is also transferred from an output port to an input port when a connection between the two is made, thus providing better behaviour of the system during run-time modification. Ports can also be reset, making the port signal that there is no data available.

Triggering of Components. When an input port receives a trigger signal, it becomes active. Every input port has an attribute called *activation type* which defines how the state of the port affects activation of component execution. There are three activation types:
(i) *Trigger*. A component is activated if *all* input ports with activation type set to *trigger* are active.
(ii) *Priority trigger*. A component is activated if *any* of its input port with activation type set to *priority trigger* is active.
(iii) *Data*. If port's activation type is set to data, it is only used to receive data, and does not affect the activation of the component.

Activation type of a port is set by the system developer and can be changed at any time to achieve the desired system behaviour. By combining these three activation types, complex triggering patterns or feedback-loops can be achieved.

As an example, component in Figure 1 has input ports *a* (data port), *b* (trigger port) and *c* (priority trigger port), and an output port *out*.

2.1. Component types

UComp distinguishes between *device components* and *software components*.

Device components. Device components represent hardware (physical) and virtual (realized using software applications) network devices. They are the base for accomplishing the uniform treatment of hardware and software components of a system. Each device corresponds to one or more device components that together cover full functionality of the device.

Device components, together with their input and output ports, are automatically generated by the UComp framework using device descriptions. Automatic generation of device components with their ports eliminates the need for specialized drivers or manual configuration of such components. Also, this allows the application of UComp to already existing components and systems, as it requires only appending them appropriate device descriptions. The input and output ports are created according to data that the device requires or provides. In addition, every device component has an output port named *connected* that signals if the device is available on the network.

Device components represent actions (synchronous request-response communication) or events (asynchronous sender-receiver messaging) of devices. Therefore, we have defined two types of device components: *action components* and *event components.*

Action Components are designed to wrap around synchronous action invocations or data queries of devices connected to computer network. Input and output ports of an action component are generated by the UComp framework to reflect arguments of the action the component represents. Action components have an additional input port *trigger* which can be used for some specific triggering patterns. When an action component is triggered, an action invocation message is sent to the device represented by this component.

Event Components allow receiving asynchronous messages from devices. These messages may signal data changes or other events that device may provide. Interfaces of event components have only output ports. These ports reflect data items that a device provides in its notification messages.

Software components. Functionality of software components is fully implemented by program code, and they are not bound to elements on the network. Some of the roles of software components are to process data, manipulate the execution of components (e.g. generation of periodical triggers), data flow control and definition of constants. Their function can vary from very simple (e.g. addition of two numbers) to complex

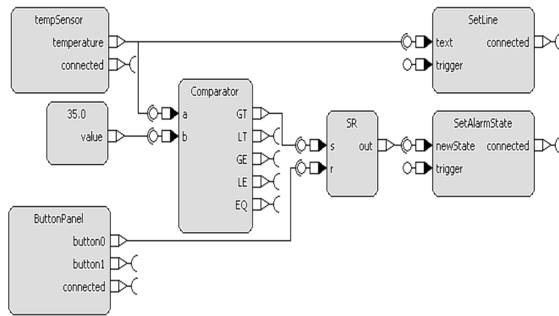

Figure 2. Graphical representation of the temperature monitoring system.

data processing. This makes it unnecessary to write any glue-code when connecting components.

2.2. Execution Semantics

Initially, all components in the system are in an idle state waiting to be activated for execution. Activation can be caused either by the triggering signals received at the input ports of the component (passive components), or by its internal events (active components). Once activated the component starts its *read-execute-write* sequence: First, the component reads all values from its input ports and stores them internally, then it executes its functionality, and finally, the component updates the values of its output ports.

Action components and most software components are passive, meaning that they execute only when they are triggered by signals received from other components, while event components and some software components are active and thus may start their execution by an internal event.

2.3. Modeling example

As an example of a system with interleaved hardware and software we will take a simple greenhouse temperature monitoring system. It consists of a sensor that monitors the greenhouse temperature, a display showing the current temperature, an alarm that sounds if the temperature exceeds 35°C, and a button that is used to acknowledge an alarm and reset it.

Figure 2 shows the graphical representation of the greenhouse temperature monitoring system developed using UComp. The system consists of *tempSensor* (temperature sensor hardware device) and *ButtonPanel* (acknowledgment button virtual device) event components, *SetLine* (display virtual device) and *SetAlarmState* (alarm hardware device) action components, *constant 35*, *Comparator* and *SR* (set/reset flip-flop) software components.

3. Realisation of UComp Component Model

The UComp architecture (shown in) is realised by a Java application that implements the Universal Plug and Play (UPnP) [6] technology to manage devices available on the network. The application communicates with devices through an UPnP control point implemented using CyberLink UPnP stack [2]. This centralized architecture has a number of benefits:
(i) It eliminates the need to change embedded device behaviour to adapt it to specific system requirements.
(ii) Embedded devices do not need to implement UPnP control points.
(iii) Run-time modification of systems is much easier.

3.1. The Middleware Layer

UPnP [6] is an open standard enables discovery, description and cooperation of devices using standard TCP/IP network protocols and technologies.

The UPnP architecture defines two types of entities: *devices* and *control points*. Devices are entities of UPnP network that provide services. Each service defines actions that are used to control the device, and state variables which model the state of the device. Control points invoke actions and/or monitor values of state variables of UPnP devices.

One of the main benefits of UPnP is the use of standards such as HTTP and XML is that it makes UPnP easily extendable. Also, UPnP is platform, language and media independent.

The fact that every UPnP device includes a full description of itself enables us to treat these devices as black-boxes, with no need for additional documentation in order to be used.

Realizing Device Components with UPnP. As UPnP supports action invocation, event notification and device description, it fits well our component

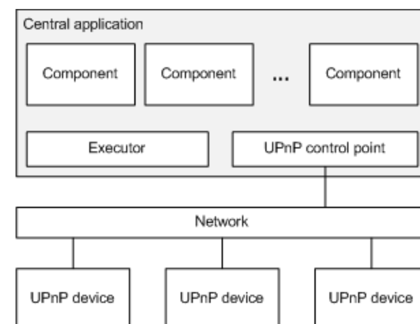

Figure 3. The UComp architecture.

technology. For every UPnP device we define a set of device components: one event component for each service that the device provides and one action component for each action defined by each service.

3.2. UComp Run-time Framework

Execution of passive components is handled by a part of the UComp framework called *Executor*. The Executor manages a queue of components that need to be executed. When a component is triggered, it adds itself to this queue. Executor sequentially takes components from the queue and calls their execute methods. At the end of a component's execute method all input triggers are reset.

Execution of active components is not managed by the Executor, but by the UPnP control stack or the components themselves.

3.3. Component Repository

Software components are stored as Java class files. This makes the creation of a component repository fairly simple. For a new component to be available deployment, it only needs to be copied to adequate directory of the file system.

Although not implemented yet, we have envisioned creation of a repository for device components. This could be realised by storing XML descriptions of known devices in a well organised directory structure.

3.4. UComp Development Environment

To facilitate the development we have created a tool for visual development of UComp systems (*UComp Developer*) and a tool for deploying them (*UComp Deployer*) to any platform that supports Standard Edition Java (*Java SE*).

The *UComp Developer* enables browsing available device and software components, and visual component composition and setup. All modifications can be done at either design or run-time. Systems developed with this tool are saved or restored from XML files.

The *UComp Deployer* tool is a Java console application that provides only the UComp framework to an existing UComp system, without the graphical development environment.

4. Conclusion and Future Work

In this paper we have proposed a simple component-based technology for developing systems containing both embedded hardware and high level software components. We achieved this by defining a component model that allows using network devices in a component-based manner. We used a standard middleware, UPnP, for implementing these devices. The component framework that we created automatically generates components using descriptions provided by such devices. We have created a set of tools that enables browsing of available components and visual composition and deployment of systems.

As future work, system design could be enhanced by providing a device component repository in the development tool. The component model could further be improved by including functional and non-functional properties in device description.. Using those properties we could perform an analysis of the system both at the design and run time. Another plan for the future is to introduce component hierarchy by adding composite components to the model.

5. Acknowledgement

Our sincere acknowledgement to Ivica Crnković from Mälardalen University for the suggestions, guidance and support provided.

This work was partially supported by the Swedish Foundation for Strategic Research via the PROGRESS research center, and the Unity Through Knowledge Fund supported by the Croatian Government and the World Bank via the DICES project.

6. References

[1] M. Åkerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pettersson, and M. Tivoli. The SAVE approach to component-based development of vehicular systems. Journal of Systems and Software, 80(5):655–667, May 2007

[2] S. Konno, Cyberlink for Java, http://www.cybergarage.org/

[3] Microsoft, .NET, http://www.microsoft.com/net/

[4] OPC Foundation, .OPC, OLE for Process Control,. Report v1.0, OPC Standards Collection, 1998, http://opcfoundation.org

[5] D. Rogerson. Inside COM. Microsoft Press, 1997

[6] UPnP Forum, UPnP Device Architecture 1.0, http://www.upnp.org/resources/documents/

[7] R. van Ommering, F. van der Linden, and J. Kramer, The Koala component model for consumer electronics software. In IEEE Computer, pages 78–85, IEEE, March 2000

[8] AUTOSAR Development Partnership, AUTOSAR – Technical Overview v2.2.1, 2008, Available at http://www.autosar.org/download/AUTOSAR_TechnicalOverview.pdf