UML 2.0 for modeling TinyOS components

Sebastian A. Bachmaier
Material Testing Institute
Universität Stuttgart
Stuttgart, Germany
sebastian.bachmaier@mpa.uni-stuttgart.de

***Abstract*— TinyOS' software architecture is based on the concept of components. These components are visualized graphically in many papers and documentation manuals. However, software architects and authors use their own graphical representation, usually. This makes it hard to understand new software quickly. The Unified Modeling Language (UML) also knows the concept of components. Although usually used in an object-oriented context, the notion is more general and UML can be used for the design of software architectures that do not base on the object-orientation paradigm. UML is nowadays known to most software engineers. The use of component diagrams as specified in UML 2.0 is therefore proposed for modeling TinyOS components, to facilitate the exchange of TinyOS software designs.**

***Index Terms*—UML 2.0, component diagram, TinyOS best practice, modeling, software architecture**

I. Introduction

The Unified Modeling Language (UML, [1]) is a standardized notation for modeling software architectures, usually used in the object oriented domain. UML includes therefore all graphical elements to model object oriented software. However, as UML is not a software development process but a notation only, it can be used in non-object oriented software development and in general in other engineering disciplines as well. UML consists of graphical notation elements and of diagram types that can be constructed from the notation elements. The graphical elements carry semantics in themselves but further meaning can be added by special notation elements, like stereotypes. UML comprises many traditional diagram types like state machines, timing diagrams and sequence diagrams, that have been in use in engineering sciences long before. However, with UML the notation is now standardized and therefore more accessible for someone who worked with UML before.

II. TinyOS and NesC Components

TinyOS programs (*applications* in TinyOS nomenclature) are written in a C programming language dialect called NesC [2]. While programming languages that are usually used for embedded system programming allow dynamic allocation of memory and dynamic binding of functions, NesC does not [3]. E.g., dynamic call resolution is too complex for general use in very constrained embedded systems. Therefore, NesC supports static binding only, however with a component model to allow for quick exchange of different implementations for certain functionality. The TinyOS component model is based on interfaces, which express the intersection of functions between components. A component can either offer or use an interface. By repeated application of this mechanism, hierarchal structures are obtained. The modules implementing or using an interface are *wired* by a special component, the *configuration*, which selects which user is connected to which provider.

III. Component Diagrams for TinyOS

A. Overview of currently used representations

Publications describing TinyOS designs usually refer to the component architecture of NesC. For better comprehensibility, the designs given are mostly backed by graphically representations. Examples are shown in Figure 1 to Figure 3.

In each of these figures, it can be seen, that the hierarchical provider and user structure is shown, as well as the interface between user and provider. However, the graphical representation is different for each figure.

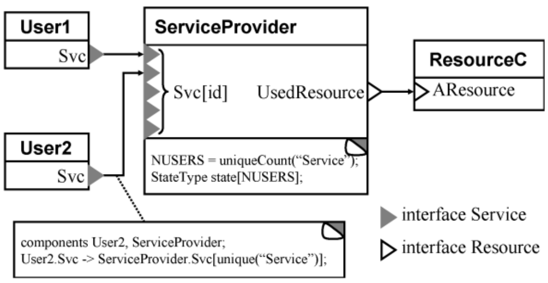

Figure 1. Component diagram as used by Culler [4]

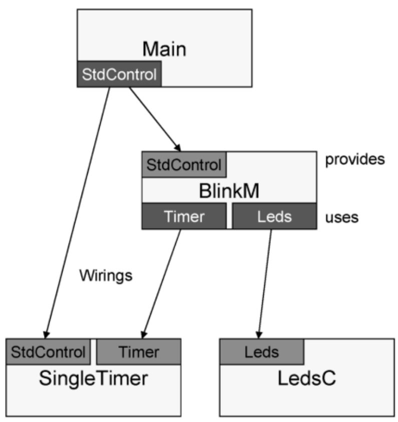

Figure 2. Component diagram as used by [5]

This work was funded by the 7th Framework Programme of the European Commission.

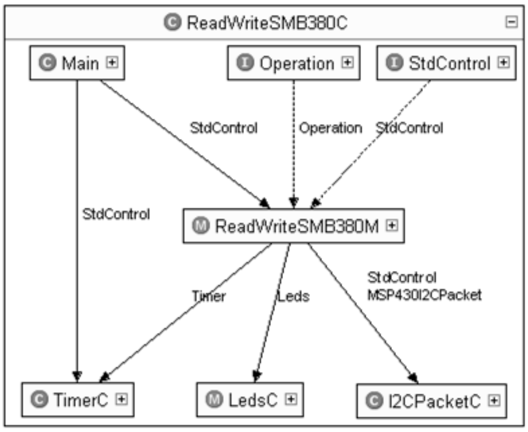

Figure 3. Component diagram as generated by the Eclipse plugins YETI [6] (and similarly YETI 2 [7])

B. *Use of UML 2*

The structure of a system, i.e. of an application, can be described by components, which represent reusable software units. According to [8], components enclose – among others – the following properties:

- a component comprises the specification of all realized and required interfaces
- a component can be exchanged by another which implements the same specification
- internals of a component are hidden, i.e. functionality of components is to be accessed via the offered interfaces only.

These three main properties are fulfilled for TinyOS components.

Components can have black-box representations, i.e. their internal implementation is not of interest, or white-box representations, where the realization is given, e.g. by means of nested components or class diagrams [9]. For the proposed TinyOS modeling, the less abstract white-box notation is proposed. The components offer interfaces and ports. Components can be exchanged by other similar components, resulting in a functionally equivalent application.

Component diagrams had been part of UML since the 1.0 version. The graphical representation changed slightly with UML 2.0 with regard to the component block layout (now a simple rectangle with a small rectangle with two bars as stereotype icon in the upper right corner) and the port element. Basic building blocks of a component diagram are the rectangle, representing a component (cf. Figure 4 a), the complete circle (*ball* or *lollipop*, naming from [10]) for a provided interface (cf. Figure 4 b) and the half-circle (*socket*) for a required ("used" in NesC-nomenclature) interface (cf. Figure 4 c).

Note that not all interfaces a component provides have to be used; however, for all required interfaces a provider is necessary.

Figure 4. Building blocks of a component diagram:
a) component with name "Name" b) provided interface with interface name "IFname" c) required interface with interface name "IFname"

As an example, the Service Instance design pattern from [4] is given in Figure 5 in UML representation. For comparison, see original representation in Figure 1.

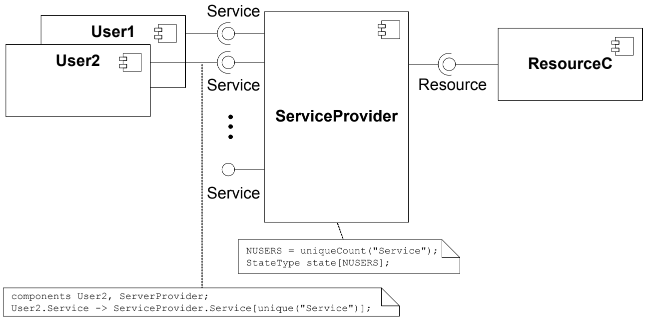

Figure 5: Service Instance pattern [4] in UML notation

The stereotypes "specification" and "realization" can be used to represent TinyOS configuration components and module components respectively. In Figure 6 these stereotypes are used to model the BlinkC component, as given in proprietary notation in Figure 2.

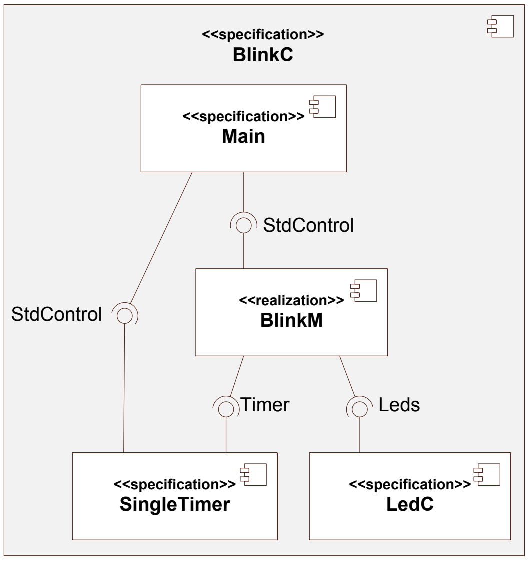

Figure 6. Component diagram in UML notation, showing use of "specification" and "realization" stereotypes

In Figure 7 the component of Figure 3 is depicted in the proposed notation, likewise giving an example of the UML *port* element. Ports assort interfaces that offer functionality together. In the suggested notation, ports offer public interfaces, which can be used by other components. Therefore, ports –

together with the *delegate* stereotype – lead interfaces through from the realizing component to the specifying component.

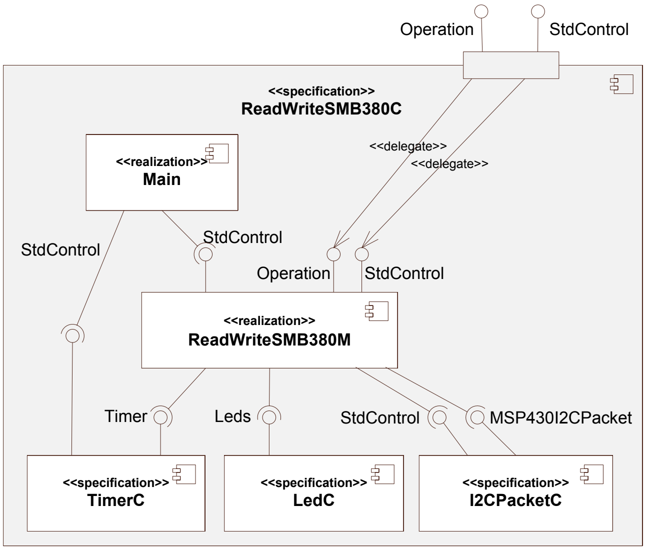

Figure 7. Component diagram in UML notation, depicting use of ports

C. Applicability in larger systems

Specifications of larger systems by use of this UML notation are possible, though they can become unclear if the diagrams are monolithic. One remedy is to model only one specification component with its realizing components in one diagram, as done in Figure 5 – 7. Between the single component view and the monolithic system view, any intermediate stage of diagram depth can be chosen, as appropriate to show the essentials of the system. Additionally, the "subsystem" stereotype can be used to model only specific parts of the whole system [11]. Using these techniques, the author was successful in specifying and documenting larger TinyOS applications.

In an automatic graphical tool, like in the YETI/YETI 2 tools, each specification component could be expanded and collapsed, thus revealing relevant and disclosing irrelevant parts of the diagram.

IV. Conclusion

The popularity of UML in computer science makes it the standard notation for documenting software architectures. The use of component diagrams is feasible and advisable to provide for a comprehensive insight to software designs, enabling efficient communication among developers and management.

The TinyOS community is invited to apply more UML and in general more design principles of the traditional PC-computer science to embedded systems. Standardization will simplify the software development for TinyOS, making design ideas more evident and finally resulting in a more economic and reusable software.

Acknowledgment

The author likes to thank Dr. Markus Krüger, Prof. Christian Große and the team of department "Non-destructive Testing and Monitoring Techniques" for their support.

References

[1] OMG Unified Modeling Language, Superstructure, V 2.1.2. Object Management Group Inc., 2007. Available: http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF

[2] (no author given). http://nescc.sourceforge.net/, date of access: 2007-10-15

[3] P. Levis. (2006). *TinyOS Programming*. http://csl.stanford.edu/~pal, date of access: 2007-12-17

[4] D. Gay, P. Levis, D. Culler. (2005, July). Software design patterns for TinyOS. In *Proceedings of the 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems.* SESSION: System design issues table of contents. pp. 40-49

[5] A. Lachenmann. (2007). *Einführung in TinyOS und nesC* (in German). http://www.ipvs.uni-stuttgart.de/abteilungen/vs/lehre/lehrveranstaltungen/uebungen/SS07/UESN_termine/start, date of access: 2007-10-15

[6] R. Schuler, N. Burri. (2006). *TinyOS Plugin for Eclipse*, http://dcg.ethz.ch/~rschuler/OLD/index.htm, date of access: 2007-10-09

[7] Benjamin Sigg. (2008). *TinyOS 2 Plugin for Eclipse*, http://tos-ide.ethz.ch/wiki/index.php, date of access: 2008-10-09

[8] M. Born, E. Holz, O. Kath. (2004). *Sotwareentwicklung mit UML 2* (in German). Addison-Wesley

[9] P. Forbrig. (2007). *Objektorientierte Softwareentwicklung mit UML* (in German). 3rd ed., München: Carl Hanser

[10] S. W. Ambler. (2004). The Object Primer: Agile Model-Driven Development with UML 2.0, 3rd ed., Cambridge University Press

[11] D. Pilone, N. Pitman. (2005) *UML 2.0 in a Nutshell*. O'Reilly

[12] B. Oestereich (2005). *Analyse und Design mit UML 2* (in German). 7th ed., München: Oldenbourg

[13] http://www.tinyos.net/tinyos-2.x/doc/html/tutorial/lesson1.html, date of access: 2007-10-15