A SystemC-centric Approach for Simulation and Generation of WSN Applications Targeted to ZigBee

F. Fummi†, G. Perbellini‡, D. Quaglia†, S. Vinco†
†Dep. of Computer Science - University of Verona, Italy
‡EDALab - Networked Embedded Systems, Italy

I. INTRODUCTION

Ambient intelligence and ubiquitous computing are the center of a great deal of attention because of their promise to bring benefits for end-users, higher revenues for manufacturers and new challenges for researchers [1]. The key aspects of these applications are their distributed nature and the presence of very limited HW resources, as in case of WSNs. Their wide adoption requires interoperability across different manufacturers, simplification of application development, simulation tools for functional validation and the fulfilment of tight HW/SW constraints.

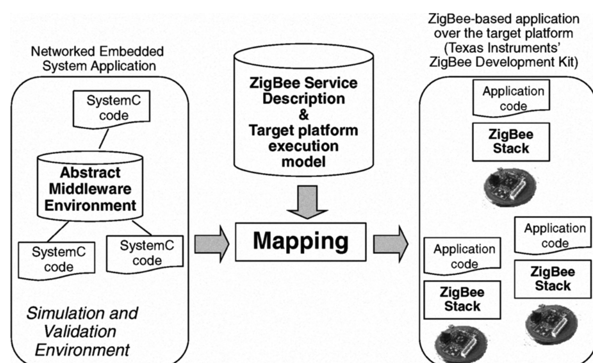

Fig. 1. Mapping of the AME-based application onto ZigBee.

The literature does not report a complete design methodology for WSN applications integrating all these aspects. The proposed methodology allows programmers to write WSN applications by using the system description language SystemC and the Abstract Middleware Environment (AME) framework for fast simulation (leftmost side of Figure 1). AME behaves as an abstraction of the services provided by the actual platform, e.g., ZigBee profiles. The SystemC framework allows to model and simulate concurrent processing, synchronization, and communication. Finally, the implemented application is automatically mapped over an actual platform, e.g., ZigBee nodes by Texas Instruments (rightmost side of Figure 1). Main advantages of the proposed methodology are: (i) *abstraction*: of the actual middleware peculiarities on AME to rapidly simulate and validate WSN applications; (ii) *automatic mapping*: of WSN applications from AME to the actual WSN platform.

II. ABSTRACT MIDDLEWARE ENVIRONMENT

Abstract Middleware Environment (AME) is a SystemC framework to write applications using a set of communication services (Abstract Middleware) and following well-known programming paradigms (i.e., Object-Oriented, Tuplespace, Message-Oriented and Database). This environment allows to design WSN applications through three steps: the first step (named AME System View (SYSV)) simulates and validates application functional requirements by using middleware-like services with different programming paradigms. During the second step (AME Network View (NWV)), a simulated network infrastructure is involved in the whole framework. AME NWV provides the same API of the previous step, even if opportunely modified to establish a communication with a network simulator. At the third step (called AME Platform View(PLV)) HW/SW partitioning is applied to each node to map functionalities to HW and SW components according to several constraints (e.g., performance, cost, and component availability). Communication is provided by AME PLV services through HW/SW/network simulators.

In this work we chose the object-oriented programming paradigm for its spread among programmers. A typical object-oriented middleware [2] provides: a mechanism to describe an object interface and to map it onto an actual object; a public repository in which instances of actual objects are registered, so that a client can obtain a local reference of a remote object; a protocol to remotely invoke object's methods with transmission of parameters and results. Let `name` and `obj` be the public name of a given object and its actual implementation, respectively. Object-oriented AME provides the following services: `registerobj(obj,name)` to register an instance of the actual object into the public repository with the given public name; `lookup(name)` to obtain a local object reference of the remote instance.

III. MAPPING ONTO ZIGBEE/Z-STACK

The mapping process consists of three key concepts:
Compliance with Z-Stack: Mapping of AME code onto a Z-Stack application might seems straightforward: initialization code can be put in the `Init()` function, while the core of the application code becomes the `ProcessEvent()` function. Actually, a problem arises with blocking calls such as the

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6934
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6934

`lookup()` service and remote method invocations: they need to stop the calling process until a response arrives. The Z-Stack operating system does not support pre-emption, thus blocking calls interrupt the event listener and prevent the processing of system events. The basic idea to solve this problem consists in replacing each blocking call by a loop in which its non-blocking version is periodically invoked. We rely on finite state machines (FSM): code statements are assigned to transitions and states are used to remember which function has to be periodically polled. Therefore, the AME-based application is converted into an FSM scheduled by using a timer: when the timer expires then a `TIMER_EVENT` is generated by the OS and the FSM re-starts from the last state reached.

ZigBee OOM Services: Once the application becomes really distributed, object references used in the AME environment have to be replaced by addresses of the nodes where the objects reside. Therefore, the public repository must link the public name of an object and its node location. The AME services are implemented on the ZigBee stack as described in the following. The *register service* is implemented by sending a message to the coordinator with the object public name and the node address, plus indicating that this is a register request. On receipt, the coordinator adds the new entry to the public repository and the new information becomes available to the other nodes. The *lookup service* is implemented by sending a lookup message containing the public name of the remote object, indicating that this is a lookup request. On receipt, the coordinator sends back a message containing the address of the node. In this way, the caller obtains the node address and can interact with it directly. The implementation of *remote method invocation* depends on the support of ZigBee Profiles. Two mechanisms are possible.

Non-profile-enabled platform: Node A wants to invoke a method on node B. A sends a message to B containing the name of the method and the parameters, plus indicating that this is a method invocation message. On receipt, node B executes the method. Then node B sends to A the parameters received and the execution result, plus indicating that this is a method execution result message. On receipt, node A obtains the result of its invocation and the application flow goes on.

Profile-enabled platform: If the target platform supports Profiles, the application can use some advanced communication mechanisms. If method invocations refer to objects supported by the profile, they can be implemented by native profile mechanisms without the need of additional code.

SystemC to C language conversion: Figure 2 shows the three steps performed to transform the AME application (written in SystemC language) into a Z-Stack compliant one (written in C language): (i) The AME application is automatically translated into an intermediate format which captures the basic syntax elements of the source code. (ii) The intermediate representation is manipulated to separate initialization statements from application statements; extract the application FSM; identify remote method invocations and calls to OOM services. (iii) The intermediate representation is automatically converted into a C-language application.

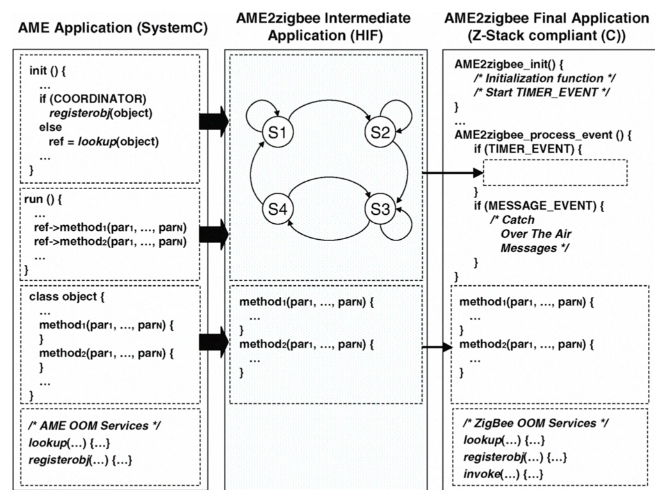

Fig. 2. AME to Z-Stack mapping process.

IV. EXPERIMENTAL ANALYSIS

The proposed SystemC-centric approach has been evaluated by considering two different ZigBee applications contained in the Texas Instruments' Z-Stack distribution, i.e., `GenericApp` and `HomeAutomation`. GenericApp establishes a connection between two nodes and periodically exchanges a string between them; HomeAutomation implements the light-control application. The former application does not use any ZigBee Application Profile while the latter uses the Home Automation Profile. AME_GA_Mapped has been generated from AME_GenericApp; AME_HA_Mapped has been generated from AME_HomeAutomation. Automatically-generated applications have been compared with the original Texas Instruments' examples and results are shown in Table I.

	Lines	Size (KB)	Transm. Overhead
GenericApp	565	111	1
AME_GA_Mapped	1261	118	2.30
HomeAutomation	720	99+100	1
AME_HA_Mapped	1590	123+105	1.02

TABLE I

RESULTS ON MAPPING EFFICIENCY.

Results show that the translation always increases the number of code lines and the binary code size but the limit of 128 KB is still satisfied. For the light-control example, binary code size has been reported for both light and switch components, respectively. Transmission overhead reveals the impact of translation on wireless communications. There is a significant difference of the transmission overhead between non-profile-enabled (2.30) and profile-enabled (1.02) applications. Without using Profiles, the emulation of the OOM programming paradigm requires more data transfers while profile-based applications already use part of these data transfers.

REFERENCES

[1] Paolo Remagnino, Gian Luca Foresti, Tim Ellis, "Ambient Intelligence: A Novel Paradigm", *Springer-Verlag Telos*, 2004.

[2] Michi Henning, "A New Approach to Object-Oriented Middleware", IEEE Internet Computing, Vol. 8, No. 1, pp. 66-75, 2004.

[3] Zigbee Alliance, "ZigBee Specification, ZigBee Document 053474r13, Version 1.0", *http://www.zigbee.org*, December 1, 2006

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2009.6934
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2009.6934