ACompleteTool-ChainforDevelopingandTestingWSNApplicationswith FLEXOR

KaminiGarg†,Anna Förster†,DanielePuccinelli†,TizianoLeidi‡ andSilvia Giordano†
†NetworkingLaboratory,ISIN-DTI,UniversityofAppliedSciencesofSouthernSwitzerland
‡ICIMSI,DTI,UniversityofAppliedSciencesofSouthernSwitzerland
{kamini.garg,anna.foerster,daniele.puccinelli,tiziano.leidi, silvia.giordano}@supsi.ch

***Abstract*—In this paper, we present the complete tool-chainfor FLEXOR, a sustainable and platform independent softwarearchitecture that is optimized to support the implementation,rapid prototyping, evaluation, and testing of wireless sensornetwork applications.**

Keywords—Wireless SensorNetworks, Architecture, Tool

I.INTRODUCTION

The proper definition of efficient software architectures for Wireless Sensor Networks (WSNs) is instrumental to code reusability across different platforms, rapid prototyping, hassle-free deployment, and the overall user friendliness of the development process. Many key challenges of WSNs have already been addressed with various degrees of success, but a significant number of valid solutions have not had the broad impact they deserve. To mitigate this problem, we advocate for a sustainable, modular, and flexible software architecture that intrinsically promotes cross-platform code reuse and fast prototyping and enables the remote control and selective activation of specific modules on individual nodes at run time. In this paper we present the complete tool-chain for FLEXOR, a platform-independent software architecture for the rapid prototyping, development and testing of WSNs.The FLEXOR software architecture and its tool-chain was developed according to the following requirements:

- **Standard programming language**
- **Platform independence**
- **High level of modularization**
- **Remote function invocation support**
- **Remote component exchange without reboot**
- **Graphical support for programming, debugging and deployment**

The remainder of this paper is organized as follows: Section II presents the overall structure and short overview ofFLEXOR. Section III presents the complete development anddeployment tool chain for FLEXOR, consisting of visual editors, codegenerators, and the run-time FLEXOR Commander.Section IV puts our work in context and compares it to otherrelevant efforts in the community. Finally, Section V discussesthe potential of FLEXOR and its possible applications tovarious challenges in WSNs. Finally Section VI concludes thepaper.

II. FLEXOR SYSTEM ARCHITECTURE

ThemaingoaloftheFLEXORsoftwarearchitectureis tosupportmodularization,remotecallbackinvocation,remote nodemanagement,andplatformindependence. Theoverall structureof FLEXORispresentedinFigure1.Themain componentsofFLEXORaredescribedasfollows:

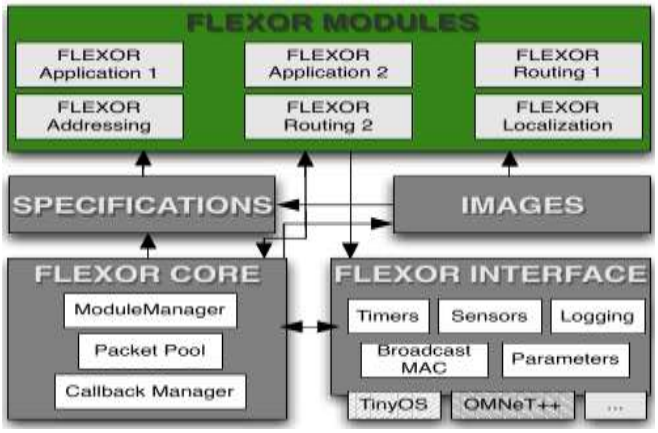

Fig.1.FLEXORoverallstructure.

A. FLEXOR Interface

FLEXORinterfaceisthemaincomponent thatenablesthe platform-independentimplementationofdifferentmodulesand alignsdifferentoperatingsystems,platforms, andevennet- work simulatorstothesameWSN-specificinterface.Currently wehaveimplementedthis FLEXORinterfacefortheTinyOS operatingsystemandtheOMNeT++Simulator.

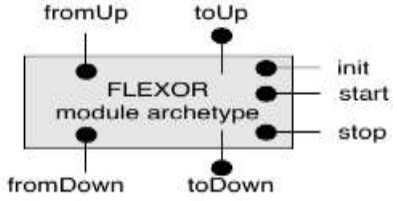

Fig.2.FLEXOR basic module archetype.

B. FLEXOR Module

A FLEXOR moduleis the basicbuilding blockfor all applicationsdevelopedin FLEXOR.Figure2presents their basicminimalstructure. Thebasicarchetype ofaFLEXOR moduleconsistsof7interfacefunctions: *init*,*start*,*stop*, *fromUp*,*fromDown*,*toUp*,*toDown*.

C. FLEXOR Core

FLEXORCoreconsistsofCallback Manager,PacketPool andModuleManager.ThePacketPoolisacentralcompo- nentin FLEXOR andservesseveralgoals.Itenablesfull controloverthenumberofmessagescurrentlyprocessedin the system, thus preventing memory overflows.The Call-backManager takescareofremotefunction callmechanism. TheModuleManageristhemostimportantcomponent of FLEXOR.It is theonlycomponentinFLEXORthatis aware of the modulescurrentlyloadedintothe systemand their organization.Itcanalsochangethisorganizationatrun-time.

D. FLEXOR Specifications and Images

Specifications consistofa setofmodulesandtheirinterconnectionsintoastack.Thestackcanbetraditional andlinear, butalsotwo-dimensional,depending ontheuserrequirements andonthemodulearchetypes used.Severalspecificationscan coexistonthesamenodeatrun-time, butonlyonesingle specification canbeactiveatanygiventime.Asetofco- existing specificationsresiding together in the memory of asinglenodeiscalledanimage.Examplesoftwoimages arepresentedinFigure3.Themodulemanagertakescare of exchangingspecificationsonruntimeasaconsequence of aninternal or external command(callback). Forfurther informationregarding FLEXORarchitecture, pleasereferto [5].

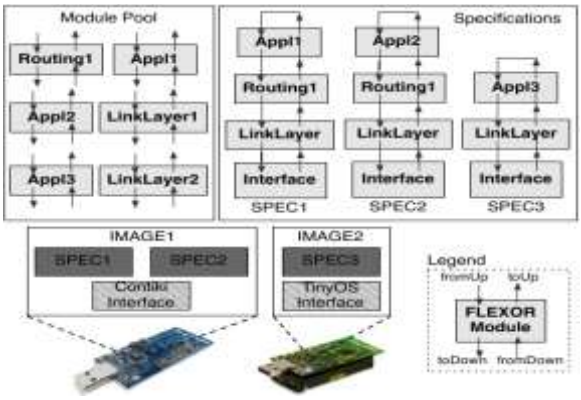

Fig.3.FLEXORspecification and image examples.

III. FLEXOR TOOL-CHAIN

The FLEXOR development environment consists of development tools for prototyping, refining, and customizing sensor network applications based on the FLEXOR software architecture. The FLEXOR environment is integrated into the Eclipse development platform [www.eclipse.org] and combines a visual domain-specific language (DSL)[7] with C source code generators and functionalities to assist the design and implementation of sensor network applications. FLEXOR is based on the Eclipse infrastructure for modeling and code generation [www.eclipse.org/modeling], and is compatible with all the Eclipse plug-ins for C and C++ development [www.eclipse.org/cdt]. Furthermore, we provide a FLEXORCommander that is able to send callback commands to any node in the network and to receive debugging information from serially connected nodes.

The FLEXOR development environment is based onmodel-driven generative programming techniques [10]. Devel- opers may design applications using a visual abstract language and generate the implementation code of the corresponding application directly from the designed software architecture. Developers may then program specific parts of the application, like processing algorithms, directly in C. By means of round- trip development support, it is then possible to extract hand- made code from the generated code and integrate snippets in the abstract design of the application. Therefore, further code generations will automatically produce the complete code of the application. Coverage of the whole development process, from application design to its refinement and optimization, is therefore supported by the tools, in an integrated and automated way (see figure 4).

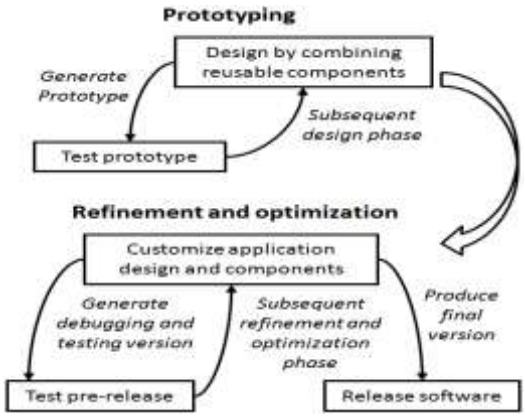

Fig.4.Development Process.

The FLEXOR development environment is composed of (see also figure 5):

• An abstraction (the model): it provides the high-level architecture of FLEXOR elements (e.g. modules, specs and images) in an editable form. It further provides rules and constraints to combine these elements. Model elements are instantiated during development and further used to design the structure and behavior of developed applications.

• Code generators: they transform the abstract information contained in the model instances into C source code. The generated code implements the structure and behavior of designed applications.

• Visual editors: they allow designers to easily instantiate, access, and modify the abstract model elements. Jointly with the model, the editors provide the visual Domain- Specific Language (DSL) that represents the centerpiece of the FLEXOR development environment.

• Support for model refactoring: it enables the modification of the model instances without modifying the behavior of the

designed applications and streamlines the whole development process.

Source code generation is used in FLEXOR to streamline the development process and reduce the risk of structural bugs in applications. Production of the source code and integration with the FLEXOR runtime is automated, consequently reducing the development complexity. Furthermore, the source code of modules and specs is produced systematically to avoid inconsistencies and programming errors. Before generation, the description of each module and specification is validated against the presence of possible design mistakes. This functionality further increases the robustness of the developed applications.

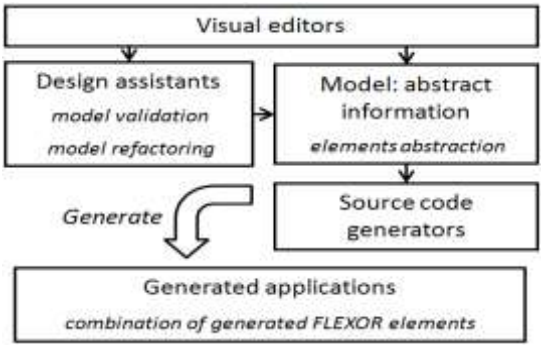

Fig.5 .Structure of the FLEXOR environment.

A. *FLEXOR abstract model*

The FLEXOR abstract model features the following main model elements (see also Section II):

- Module: it provides the information describing the external parts of a module. The module editor enables the addition of input and output access points that the module uses to communicate with the other modules. Any number of inputs and outputs may be added. To ease the design of modules, it is possible to specify an FLEXOR module, whose properties are reused and may be extended by the module. FLEXOR types are a light-weight form of programming inheritance, enabled through encapsulation of STRUCTS that allows reuse and customization of common module designs.
- Module Implementation: it contains the abstract information of the internal parts of a module. For each module it is possible to specify a module implementation, which encapsulates all the source code snippets for functions associated with input and output access points, for initialization of functions, module callbacks and payloads.
- Specification: it allows the combination of different modules into module stacks, by connecting input and output gates.
- Image: it provides a way to aggregate more than one specification into an application, which may be deployed on sensor nodes.

Further model elements are available in the FLEXOR abstract model for payloads and callbacks. They may be used to specify code identifiers and special purpose state variables. For each model element a visual editor or a form editor is availablein the FLEXOR development environment. A visual editor is provided for modules and specifications (see figure 5). A form editor is provided for module implementations and images, which mostly store information in the form of text or as references to other model elements.

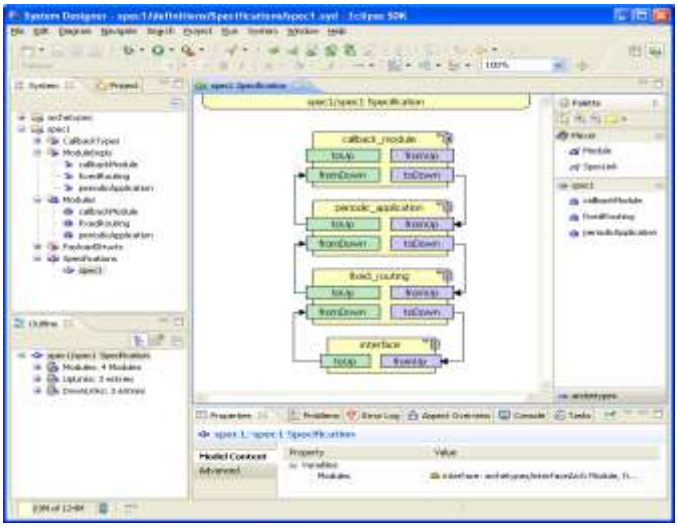

Fig.6 .FLEXOR graphical user interface.

B. *Code generators and generated code*

FLEXORsourcecodegenerators queryFLEXORabstract modelelementsandproduceCsourcefiles. Generatorsperformatransformation fromanabstractDSLtoastructured programminglanguage,bycustomizing theFLEXORsoftwarearchitectureforeachspecific applicationdesignedwith theFLEXORdevelopmentenvironment.Thefollowingcode isgenerated:

- Source files are generated for each module and associated module implementation, which defines a C STRUCT containing all the internal state variables of the module and pointer to functions for input and output gates used for inter-module communication. Thegenerated source files further contain initialization functions used to set default values of the internal state and to properly install the required function pointers.
- For each specification, a source file with a C STRUCT containing the STRUCTS of all modules used in the specification is generated. Initialization functions that allow different modules to communicate are also available. Such functions are used to properly install the involved function pointers on connected modules.
- For each image a source file with a C STRUCT containing the STRUCTS of all the required specifications is generated. Furthermore, source code to initialize and configure the generated application is produced.

Generated source code files extend and specialize the basicfunctionality provided by the FLEXOR core. Furthermore, all needed components (modules, specifications, payloads, callbacks) are included into the

system to be compiled au- tomatically, thus increasing further the user-friendliness of FLEXOR.

Fig.7. FLEXOR commander enables the end user to interact with any node in the deployed network.

C. FLEXOR Commander

As a final component of our FLEXOR tool-chain we present the FLEXOR interaction server, see Figure7. This GUI-supported tool enables the end user of the sensornet deployment to communicate to the nodes in the network via callbacks. Here, the user can send FLEXOR packets via serial port to one or more serially connected nodes, which then forward the callback to its final destination. The server also displays debugging information from serially connected nodes and can flash them with new code.

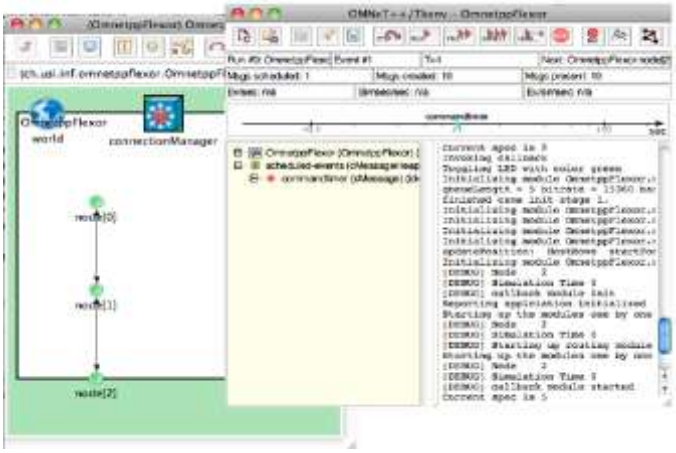

Fig. 8. Screenshot of FLEXOR on OMNET++ Simulator.

D. FLEXOR on OMNET++ Simulator

In order to validate and verify the WSN applications, we have also ported FLEXOR on OMNET++ platform. Porting FLEXOR on OMNET++ platform helps the developers to debug their applications before real deployment. Therefore after validation and error correction of a WSN application, the code is linked to a real hardware platform such as TinyOS that is already implemented for FLEXOR. Figure 8 depicts the sample working of FLEXOR on OMNET++ platform.

IV. RELATED WORKS

In this section, we contextualize FLEXOR with respect to other related efforts in the WSN community. In essence, FLEXOR combines the best practices and ideas into a single architecture at the price of a low overhead.

A. Software architectures

The importance of a sustainable WSN architecture is often emphasized in the literature [3]. Merlin et al. [9] have recently identified a set of properties that need to be supported by WSN software architectures: modularity, universality, event notification, service support, and information propagation.

In terms of modularity, universality, and flexibility, X-Lisa, SNA, Chameleon, and FLEXOR have very similar properties, even if the details are different. However, FLEXOR offers the most flexibility, as it allows for any number and order of modules in its stack and even for two-dimensional stacks. Chameleon's [4] architecture is very modular due to its use of Rime, a lightweight layered communication stack for sensor networks that implements different communication primitives (e.g. over 802.15.4, IP, etc.), but it fails to enforce any modularization at the application level. FLEXOR, on the other hand, enforces modularization at all levels and imple- ments services and functions only at the medium access layer (one-hop unreliable broadcast) along with platform-abstraction functionalities.

B. Virtual machines, code distributions and remote control

Traditional virtual machines like Darjeeling [1] or Mate [8] have different goals, as they do not aim to provide a sustainable architecture for WSNs. Although their main goal is the re- programming of motes after deployment, they only enable the full exchange of the complete code at a node, as opposed to partial exchanges. However, they do not enable modulariza- tion, re-usability of code, or very low-overhead runtime soft- ware management. Additionally, they do not enable software management from inside the network: e.g., a node may not drive its own software components or the ones of its neighbors.

C. Graphical user interfaces and development enviornments

Various languages and modeling environments have been also proposed for specific operating systems, mainly for TinyOS. DSN [2], a declarative language for WSNs, or the Gratis [11]: modeling environment, are such examples. How- ever, while usually offering a higher level of abstraction and a more user-friendly implementation environment, they are not targeted towards code reuse, modularization or software component management.

V. THE POTENTIAL OF FLEXOR

FLEXOR can be used in many different applications and deployments. We have already applied FLEXOR to the im- plementation of various link layer and routing protocols, to the collection and management of link level statistics from infrastructure-less testbeds, and to the management of mobile nodes in WSN testbeds. In the next paragraphs, we briefly describe these implementations and also sketch some other advanced applications and features of FLEXOR.

Mobile testbed nodes: Managing mobility in testbeds and deployments is a major challenge, as nodes typically rely on a wired backchannel to receive new software, log data, etc. With FLEXOR, these features can be taken over by the callback invocation and the run-time management of modules, as described in Section II.

Link level traces: Link level traces represent an important tool to boost the realism of simulation. It is critical to be able to collect such traces not only from traditional backchannel-based testbeds, such as MoteLab [13], but in any environment, including outdoors. FLEXOR simplifies this task by making it possible to easily send commands to the nodes, change their behavior, and even collect the traces at a centralized sink point without the use of a backchannel.

Evaluation of communication protocols and services: Another typical task for WSN developers is to protocol or service evaluation and benchmarking. Typically, protocols are loaded and tested in a sequence. FLEXOR simplifies this task, especially in a real-world deployment where backchannels are not available. Callback invocation is used to easily exchange protocols without affecting the state of the other protocols.

Fairness and visibility: Fairness is a major challengewhen several protocols co-exist on the same node [6]. Pay- loads coming from different modules are packed into the same packet, thus minimizing the overall network traffic. The extreme modularity of FLEXOR allows for better visibility of the individual modules [12], as individual modules can be clearly separated and their internal state and processes can be logged.

Software rejuvenation:FLEXOR enables the long-termmanagement of software modules, known as software reju- venation [14]. Software rejuvenation can be easily achieved with FLEXOR by using the module interface and its main functions. Instead of rebooting a node and loosing its complete state, all of the FLEXOR modules can be re-started at any time and thus achieve rejuvenation of the existing state or a different secure specification can be loaded to backup the node state.

Local and remote debugging: FLEXOR can also beused together with any platform-dependent and independent debuggers, code inspections, and visualization mechanisms, since it is entirely C-based. As discussed before, we also have our FLEXOR Commander to do such a task.

Cross-layer support:FLEXOR allows a great deal offlexibility in the definition and use of its module stack, as it enables two-dimensional stacks as well as cross-layer communication. This can be achieved in two different ways: adding new inputs and outputs to the modules to connect non-neighboring modules, and event notification, enabled via callback invocation. This is a very important feature for FLEXOR, as many WSN optimization techniques rely on cross-layer communication and control.

VI. CONCLUSION

We have presented the design and complete tool-chain ofFLEXOR, a modular and flexible software architecture forthe rapid prototyping of WSNs. FLEXOR lowers the barriers to entry into the traditionally challenging WSN development process by offering a platform-independent software archi- tecture as well as a user-friendly programming environment and toolchain. FLEXOR represents an orthogonal effort with widely used WSN operating systems such as TinyOS and Contiki. FLEXOR can also be viewed as a framework for the integration of advanced debugging techniques such as passive in-field inspection of WSNs. FLEXOR has the potential to streamline WSN development by encouraging code reuse. As part of our follow-up work, we plan to implement a large number of components to enrich FLEXOR's basic set of modules.

REFERENCES

[1] N. Brouwers, K. Langendoen, and P. Corke. Darjeeling, a feature-rich vm for the resource poor. In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, SenSys'09, pages 169–182, Berkeley, California, 2009. ACM.

[2] D. Chu, L. Popa, A. Tavakoli, J. M. Hellerstein, P. Levis, S. Shenker, and I. Stoica. The design and implementation of a declarative sensor network system. In Proceedings of the 5th international conference on Embedded networked sensor systems, SenSys '07, pages 175–188, Sydney, Australia, 2007. ACM.

[3] D. Culler, P. Dutta, C. T. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, and J. Zhao.Towards a sensor network architecture: lowering the waistline. In Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10, pages 24–24, Santa Fe, NM, 2005.

[4] A. Dunkels, F. O¨ sterlind, and Z. He. An adaptive communication architecture for wireless sensor networks.In Proceedings of the5th international conference on Embedded networked sensor systems(SenSys), pages 335–349, New York, NY, USA, 2007. ACM.

[5] A. Förster, K. Garg, D. Puccinelli, and S. Giordano. Flexor: User friendly wireless sensor network development and deployment. In Proceedings of the 13th IEEEInternational Symposium on a World of Wireless, Mobile and Multimedia Networks, San Francisco, CA, USA, 2012.

[6] J. Il Choi, M. Kazandjieva, M. Jain, and P. Levis. The case for a network protocol isolation layer. In Proceedings of the 6th ACM conference on Embedded network sensor systems (SenSys), Berkeley, CA, USA, 2009.

[7] I. Kurtev, J. Be´zivin, F. Jouault, and P. Valduriez. Model-based dsl frameworks. In Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, OOPSLA '06, pages 602–616, Portland, Oregon, USA, 2006. ACM.

[8] P. Levis and D. E. Culler. Mate: A virtual machine for tiny networked sensors. In Proceedings of the ACM Conference on Architectural Support for Programming Languages and Operating Systems, 2002.

[9] C. J. Merlin, C.-H. Feng, and W. B. Heinzelman. Information-sharing protocol architectures for sensor networks: the state of the art and a new solution. SIGMOBILE Mobile Computation and Communication Revue,13:26–38, March 2010.

[10] B. Selic. The pragmatics of model-driven development. IEEE Software, 20(5):19 – 25, Sep-Oct 2003.

[11] P. Vo¨ lgyesi, M. Maro´ ti, S. Do´ ra, E. Osses, and A. Le´deczi. Software composition and verification for sensor networks. Science of Computer Programming, 56:191–210, April 2005.

[12] M. Wachs, J. I. Choi, J. W. Lee, K. Srinivasan, Z. Chen, M. Jain, and P. Levis. Visibility: a new metric for protocol design. In Proceedings of the 5th international conference on Embedded networked sensor systems (SenSys), SenSys '07, pages 73–86, Sydney, Australia, 2007.

[13] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: a wireless sensor network testbed. In Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN), pages 483–488, April 2005.

[14] M. Woehrle, A. Meier, and K. Langendoen. On the potential of software rejuvenation for long-running sensor network deployments. In

Proceedings of the 1st International Workshop on Software Engineering
for Sensor Networks (SESENA), Cape Town, South Africa, 2010.