A Model of Computation for Sensor Network Application Language

Class Project for EE290N, Fall '04
Author: Alvise Bonivento
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Abstract—**In this paper we introduce a new model of computation that is the basis for SNAL: a Sensor Network Application Language. In our design flow, the user describes the network in terms of logical components, queries and services, and then SNAL captures these specifications and produces a set of requirements that the network has to satisfy to ensure a correct functionality. We describe a MoC that supports this language.**

I. INTRODUCTION

The complexity of the design task and the variety of skills needed to develop applications and implementation platforms, make the task of developing an effective design methodology for Wireless Sensor Networks (WSN) quite challenging.

The ideal solution should deal with all phases of the design process from conception to implementation. As often is the case in other application areas, methods and tools are readily available to develop hardware platforms, [1][2][3], but they lack for other phases that deal with higher level of abstractions.

The most common design methodology for WSN starts with the description of the protocol specifications using the NesC/TinyOS stack [4]. The NesC/TinyOS platform was developed at U.C.Berkeley and it leverages on a "method call" model of computation. It was designed to describe component-based architectures using a simple event-based concurrency model. This platform was enriched with a simulation environment called TOSSIM [5]. Its success is also related to the wide spreading of the hardware platforms of the Mica family [3]. Remarkably, the combination of Mica and TinyOS allowed for a great push of research in the WSN community.

Alternatively, protocol solutions are simulated using environment such as OMNET++ [6] or VisualSense [7] and then implemented in NesC/TinyOS. Omnet++ is a discrete event simulator developed by Andras Varga at the Technical University of Budapest. Although not specifically targeting the WSN domain, Omnet++ is widely used within the communication community for protocol simulations, especially in European institutions.

Visualsense is a modeling framework for WSN developed as part of the Ptolemy project at U.C.Berkeley [8]. It is an extension of a discrete-event model with an extra capability of describing properties of the wireless connectivity. Visualsense is certainly a powerful tool to model and evaluate protocol solutions under different scenarios.

Although an effort to move to a higher layer of abstraction is certainly visible, especially with Visualsense, we believe that the current design flows lack of a top-down approach.

In [9] it is proposed a universal application interface, which allows programmers to develop applications without having to know unnecessary details of the underlying communication platform, such as air interface and network topology. The goal of [9] was "defining a standard set of services and interface primitives (called the Sensor Network Services Platform or SNSP) to be made available to an application programmer independently on their implementation on any present and future sensor network platform". Motivated by this work, we decided to implement a Sensor Network Application Language (SNAL).

The design flow that we envision can be summarized as follows:

1. Application is specified using SNAL and constraints for the network architecture are generated
2. A protocol stack with an abstraction of a hardware platform (network architecture) is selected as a solution of a covering problem. This implies the creation of a library of network architectures whose performance can be modeled using protocol level tools (i.e. VisualSense or Omnet++)
3. Once the protocol stack is selected, a hardware platform exploration can be performed. Metropolis is a candidate for this step. Alternatively, if the hardware platform is given (i.e. Mica, TELOS) the protocol stack can be directly implemented using specific tools (i.e. NesC/TinyOS)

The goals of SNAL should be, in order of priority:

1. Allow user to describe the network in terms of logical components queries and services, as suggested in [9], without any knowledge of the protocol stack or the hardware platform.
2. Capture these specifications and produce a set of constraints that the protocol stack and the hardware platform have to satisfy to ensure correct functionality of the network
3. Simulate WSN applications whenever an abstraction of the protocol stack and the hardware platform is available.

We believe that SNAL should have the following characteristics:

1. It must be component based, and create a logical architecture in the WSN where controllers are masters and sensors and actuators are slaves, as suggested in [9]. This will allow for exploiting the specificity of our domain.

2. Since SNAL is based on a service platform it should have a “publish/subscribe” flavor to express the access at these services.
3. It must capture all the possible scenarios that a given program can generate.

A language is a combination of a MoC and a set of primitives. In [9] some primitives for SNAL are described, but there is no indication of what a pertinent MoC should be.
In this paper we present a MoC to support SNAL. First, we explain the features of our MoC using an intuitive approach. Secondly, we provide a formal description using an extension of the Tagged Signal Model (TSM) [10] notation.
Although only field examples can give a final word, we believe that our solution offers a good trade-off between the necessary power of expressiveness and analysis capability typical of domain specific languages.

II. The MoC

According to [9] a WSN is composed of three types of logical components, Virtual Controller (VC), Virtual Sensor (VS), Virtual Actuator (VA), and a set of services. Specifically, [9] identifies six types of services: two of them, Query service and Command service, are used by the logical components to communicate, the other four, Time Synchronization Service (TSS), Location Service (LS), Resource Management Service (RMS), and Concept Repository Service (CRS), are used by the logical components to interpret the content of the received queries or commands.
Consequently, in our MoC we consider only four types of components: the three logical components VC, VS, VA, and a service component CRS that groups all the remaining services.
Each logical component can call the CRS (and this is the publish/subscribe part of our MoC). The only connections allowed between logical components are the ones between: VC and VC, VC and VS, VC and VA. Consequently, no connection is allowed between two VS’s (that would be a multihop routing and as such a protocol choice) and no direct connection between VS and VA.

1. Virtual Controller

A VC is an abstraction of a set of controlling algorithms for the WSN. Ideally, the VC runs a controlling algorithm and when it needs data or decides an action it sends a query or a command to another logical component. Since the VC is the “brain” of a WSN it has to be a powerful actor. Consequently the only restriction to its behavior is a simple causal relationship between the events of sending queries and the events of receiving a reply.

2. Virtual Sensor

A VS is an abstraction of a set of measuring devices and it is defined by the list of parameters that can be read, and by the primitives that are used for reading them. At this level we are not interested whether it is implemented by a sensor, by a set of sensors, or by a controller with sensing capabilities.
The VS works as follows:

- It reads a query from an input channel and asks the CRS for interpretation of the query
- After receiving that interpretation, the VS advances its thread with the requested parameters and produces the data as required.
- The VC responds to the query.

An interesting situation happens when more than one VC is sending queries to the same VS. For example: a VS has two input channels from two different VC’s. From channel 1 it receives a query that is interpreted as “give me humidity measurements until time T1 with a particular sampling rate R1” and it advances its thread accordingly. Then, from the same input channel receives another query that can be interpreted as “give me humidity measurements until time T2 with a sampling rate R1” and advances accordingly. Then, from the other input it receives a query that means “give me humidity measurements until time T3 with a sampling rate R3” and assume $T1<T3<T2$ and $R1<R3$. Consequently, to service the third query, the VS should backtrack and reproduce the sampling with a higher rate. This happened because we are using an untimed MoC, but the content of the queries carries some information on the real time that is captured by the CRS.
A clean way to deal with this problem is to introduce a blocking read for the VS. Reviewing the previous example, after receiving the first query, the VS waits for the other input channel to have a query and only when both inputs have a token it evaluates the two queries together. As a result the VS advances its sampling process with rate R3 (the highest of the two) until T1 (the earliest deadline). At this point the first query is fully serviced and the token on the first input is consumed. The other query has not been completely serviced yet, hence the relative token remains. When another query arrives at the first input channel the query evaluation process starts again, the VS advances and so on.
When a VC does not need to communicate to the VS anymore, it sends a t symbol. The meaning of the t symbol is "any behavior from now on". Consequently, such a query will never be completed and the relative token remains there. This way we do not introduce artificial deadlocks. When a VS interface has t 's in all its inputs it means that it is not needed anymore and stops executing.
As a result, advancing the VS “little by little” all the specifications are captured and we are able to correctly characterize all the dynamic of the sensing requirements.
Notice that this blocking read is only done to capture the specifications at the application level and create correct requirements for the network architecture. At protocol level, when the application is mapped into real time, there is no commitment to implement the blocking read.
From an expressiveness perspective, introducing a blocking read limits the behavior of the VS to what the VC decides. This way we force a master-slave relationship and this is a way to explore the specificity of this domain.

3. Virtual Actuator

A VA is an abstraction of a set of devices that can influence the environment. The semantic is very similar to the case of the VS, with the difference that the connections between a VC and VA carry only commands, and that the replies carry only acknowledgements.

III. Extension to the Model

Assume that connections between the VC and the VS are constrained to be single port connections.

Consider the following scenario with two VC's connected to two VS's, and:

- VC1 waits for the data coming from VS1 to decide if sending or not a query to VS2
- VC2 waits for the data coming from VS2 to decide if sending or not a query to VS1

The proposed blocking read would create a deadlock. Nevertheless this is a realistic scenario we need to be able to capture (typical of an emergency reactive network where if a controller receives an alert message from a set of sensors it also queries another set of sensors to accurately characterize the issue). This is due to the fact that at some point in the VC codes there is a "if, then, else" block whose relative branches have a "send query" instruction. Since we are allowing for single connections only, the MoC fails to capture this scenario.

A simple extension to the model is allowing multiple ports connections between the VC and the VS. In particular, a connection has as many ports as the number of branches in the VC program. As a result this number does not need to be specified by the user, but it can be inferred by the branching tree of the controlling algorithm. When a particular branch does not need the services of a particular VS, it sends a t symbol in the relative connection.

This way we are able to capture all the possible scenarios of the WSN and generate a correct set of requirements.

IV. Related MoC's

In this section we try to relate the proposed MoC to other existing ones outlining similarities and differences.

The communication between logical components is similar to a Process Network with a blocking read mechanism to prevent non determinism for the virtual sensor and actuator, and without the blocking read for the virtual controller. This tiered architecture is fundamental to concentrate all the decision capabilities at the virtual controller.

As already mentioned SNAL has to support a complex tag system. With reference to the Tagged Signal Model (TSM) notation, consider a connection as a signal and a query as an event. This event has a tag (that identifies its order within the signal), and a value (content of the query). The value itself is a composition of two types of information: the type of data required and the temporal scope of those data. This last information is referred to real time and enables to order queries coming from different virtual controllers. As already explained the only entity that has a notion of this order is the CRS. Consequently, the CRS is able to order the query according to their real time and also to intersect the sensing requirement expressed in their content. We believe this is similar to the concept of unification of tag and values expressed in [11][12]. In our solution this unification is performed on demand via a publish/subscribe mechanism.

Another asset of our MoC is the capability of capturing all possible scenarios. This idea was inspired by Ulisse [13]. In Ulisse the application is described using Message Sequence Charts to capture simple scenarios, and then different scenarios are composed using a Petri Net structure. The main difference with Ulisse is that in our case we already know what type of components populates the network and we are able to characterize them in a sort of hierarchy. Leveraging on this knowledge, we are able to propose a component based approach that is able to maintain a high level of expressivity (as was the case with Ulisse), and also allows for a more synthesis oriented view of the system.

The idea of introducing the t query to resolve unwanted deadlocks can be seen as a particular case of the *null* message introduced by Misra in [14] when dealing with asynchronous parallel simulations. Similarly to [14], the t query does not have any physical implementation, but it is a useful notation to avoid deadlock when capturing specifications.

A closer look at the behavior of the logical components outlines their similarity to the threaded processes of Metropolis [15]. In particular we believe that the separation of the component in task and interface is a first important refinement. Ideally, from the task we will be able to generate the computation algorithm for the single hardware components, while from the interface we will read the requirements for the communication network. Furthermore, we believe that a Quantity Manager is a perfect candidate for the implementation of the CRS. These considerations drive our next step of integrating the proposed MoC in Metropolis.

An interesting approach to the design of interfaces for sensor network components is described in the standards of the IEEE 1451 family [16]. This family of standard was created to improve the reusability of the network and component solutions for sensor networks within manufacturing plants. Although the initial targets were wired networks, the applicability of those concepts to a wireless solution seems to be appealing. In IEEE 1451 there is a first concept of logical components, where a sensor for example identifies a group of sensing devices rather than a single hardware component. Another interesting aspect is that the interfaces for the controlling entities of the network are more powerful than the ones for sensors and actuators, therefore implying a master-slave logical architecture as we do. Nevertheless, those standards are specifically target to the design of interfaces and they can hardly be generalized to a complete application language.

V. Formalization

In this section we give a more formal characterization of the proposed MoC. Although a description using the classical TSM would be appropriate, the presence of a concrete semantics makes the description hard to read. We decided instead to describe the MoC borrowing the idea presented in

[11][12] of using an extra tag set to represent an order relation among events of different signals (variables).

In particular we define a query as an event of the type $e = (x, c, n, v, T)$, where: x is the reference variable (in our case the relative connection), c can be either 0 or 1 and represents the fact that the query was generated by a VC asking for data (c=0) or a VS replying to a previous request (c=1), n is natural number and indicates the sequence number of the query within the variable x (the internal tag), v expresses the sensing (or actuating) requirements, T is a real number specifies the end of the temporal scope of the query (reference to real time). With this notation a t query is an event of the type $e = (x, 0, n, *, \infty)$.

A VC with M connections is a process described by a set of variables (connections) $P = (x_1, .., x_M)$ and a causality relationship among elements within the same variable that can be expressed as follows: for each $e = (x, 1, n, v, T)$, there must exist an event $e' = (x, 0, m, v', T')$ with m<n (a reply follows a request).

A VS (and similarly a VA) with M connections is a process $P = (x_1, .., x_M, x_A)$, where $x_1, .., x_M$ refer to the M connections and x_A describes the advancing behavior of the VS. The connection variables have a causality requirement equivalent to the one expressed for the VC. We also have to express the blocking read and the slow advancing with intersection of the sensing (or actuating) requirements. Call the first event of x_A: $e_0 = (x_A, 1, 1, *, 0)$. For each two events of x_A of the type $e = (x_A, 1, n, v, T)$ and $e' = (x_A, 1, n+1, v', T')$, it must be $T \le T'$, and there exist M events of the type $e_i = (x_i, 0, n_i, v_i, T_i)$, $1 \le i \le M$, such that $\forall i, T \le T_i$ and $\exists i, T_i \le T'$ and $v' = \bigcap_{i=1}^{M} v_i$.

The composition of VC, VS, and VA processes can be easily obtained matching the corresponding connections. The result is a WSN process.

VI. CONCLUSIONS

We introduced a MoC to support SNAL: a Sensor Network Application Language. The initial goal of this language is to capture the specification at the application level and translate them into a set of requirements for the network architecture.

Our solution allows for specifying the application in terms of logical components (controller, sensor and actuators) and services, completely independent from the implementation space. SNAL is characterized by an untimed MoC that merges typical process network characteristics with a publish/subscribe flavor typical of application layer languages. This MoC forces the description of the WSN to have a controller centric view, limiting the behavior of sensor and actuators using blocking read mechanism.

We believe the proposed MoC allows for the right mix of expressiveness and analyzability required by such a domain specific language.

REFERENCES

[1] J. Rabaey et al., ``PicoRadio Supports Ad Hoc Ultra-low Power Wireless Networking", IEEE

[2] J. Kahn, R. Katz, and K. Pister, ``Next Century Challenges: Mobile Networking for Smart Dust", MobiCom, 1999.Computer Magazine, July 2000.

[3] J. Hill, D. Culler, ``Mica: A Wireless Platform for Deeply Embedded Networks" IEEE Micro., vol22 (6), Nov/Dec 2002, pp.12-24.

[4] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, ``The nesC Language: A Holistic Approach to Networked Embedded Systems", Proceedings of Programming Language Design and Implementation (PLDI) 2003, June 2003.

[5] P. Levis, N. Lee, M. Weksh, and D. Culler, ``TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Application",SENSYS 03.

[6] A. Varga, ``The OMNeT++ Discrete Event Simulation System", in European Simulation Multiconference June 2001.

[7] P. Baldwin, S. Kohli, E.A. Lee, X. Liu, Y. Zhao, ``Visualsense: Visual Modeling for Wireless and Sensor Network Systems", UCB ERL Memorandum UCB/ERL M04/8 April 23, 2004.

[8] http://ptolemy.eecs.berkeley.edu

[9] M. Sgroi, Adam Wolisz, Alberto Sangiovanni-Vincentelli and Jan M. Rabaey, "A Service-Based Universal Application Interface for Ad-hoc Wireless Sensor Networks", whitepaper, U.C.Berkeley 2004.

[10] E. A. Lee and A. Sangiovanni-Vincentelli, ``A Framework for Comparing Models of Computation", IEEE Transactions on CAD, 17(12), December, 1998.

[11] A. Benveniste, B. Caillaud, L.P. Carloni, P. Caspi, and A.L. Sangiovanni-Vincentelli, "Heterogeneous Reactive Systems Modeling: Capturing Causality and the Correctness of Loosely Time-Triggered Architectures (LTTA)", Proceedings of the Fourth International Conference on Embedded Software (EMSOFT), 2004

[12] A. Benveniste, L.P. Carloni, P. Caspi, and A.L. Sangiovanni-Vincentelli, "Heterogeneous Reactive Systems Modeling and Correct-by-Construction Deployment", Proceedings of the Third International Conference on Embedded Software (EMSOFT),

[13] M. Sgroi, "Platform-based Design methodologies for Communication Networks" Ph.D. Thesis, U.C.Berkeley, Fall 2002.

[14] J. Misra, "Distributed Discrete-Event Simulation".

[15] A. Pinto, "Metropolis Design Guidelines", U.C.Berkeley, UCB/ERL Memo 04/40, November, 2004

[16] http://www.motion.aptd.nist.gov/