# INTEGRATING WIRELESS SENSOR NETWORKS AND THE GRID THROUGH POP-C++

Augusto B. de Oliveira[1], Lucas F. Wanner[1], Pierre Kuonen[2] and Antônio A. Fröhlich[1]

[1] *Laboratory for Software and Hardware Integration*
*Federal University of Santa Catarina*
*PO Box 476 – 88049-900 – Florianópolis, SC, Brazil*
{*augusto, lucas, guto*}*@lisha.ufsc.br*

[2] *Grid and Ubiquitous Computing Group*
*University of Applied Sciences of Fribourg*
*PO Box 32 – CH-1705 – Fribourg, FR, Switzerland*
*pierre.kuonen@eif.ch*

**Abstract** The topic of interaction between Wireless Sensor Networks (WSNs) and other computation systems has received relatively low scientific attention, and the interface between the data source and the applications that use that data remains a problem for the application programmer. This work extends POP-C++, a programming language and runtime support system for Grid programming, to enable Grid applications to seamlessly and concurrently use WSNs' sensing and processing capabilities.

**Keywords:** WSN, Grid, Remote Objects

## 1. Introduction

Even though Wireless Sensor Networks (WSNs) have been the focus of many research effors in recent years, the topic of interaction of WSNs with other computing systems has received relatively low attention. The research efforts that do address this issue, such as TinyDB and Cougar, abstract the individual sensor nodes and give access to the WSN as a whole, allowing the applications to perform queries as they would to a database; while this level of abstraction allows for the optimization of queries, minimizing the amount of messages to be sent over the wireless link, we feel that it takes power away from the application programmer, as further exploration of the WSN nodes' processing capabilities becomes

difficult. Furthermore, such a solution does not hide the frontier between the WSN and the rest of the computing system.

In contrast to these approaches, we integrate WSNs and the Grid seamlessly without removing power from the application programmer by extending POP-C++ [Nguyen and Kuonen, 2007]. POP-C++ is a pre-existing object-oriented grid programming language and runtime support system, capable of supporting distributed, parallel objects over a network. The specific goals of our extension are to allow:

- Grid applications to communicate with WSNs seamlessly: By hiding all network interaction under a remote method call interface, details of the network stack and physical medium are transparent to the application;
- Concurrent use of the WSN sensing capabilities by multiple Grid applications: By allowing multiple objects to run on each node and each object to be used by multiple interfaces, concurrent use of the WSN by multiple applications is made possible;
- Application independent sensor node software: by allowing applications to use a common set of methods, we hope to minimize the need for costly re-programming of node program memory and allow additional applications to be initiated after the WSN is deployed.

The structure of this article is the following: In section 2 we introduce the POP-C++ programming language and runtime support system; in section 3 we detail how POP-C++ was extended into WSNs; in section 4 we evaluate our implementation; in section 5 we present the design challenges found in WSNs; section 6 presents related works that share our goal; finally, in section 7 we present our conclusions.

## 2. POP-C++

POP-C++ is an extension of C++ created to support requirement driven, distributed parallel objects. In POP-C++'s object model, parallel objects have the ability to describe their resource needs at runtime and are allocated in any of the remote nodes that can support its execution. The process of finding a suitable node and transmitting the object code is transparent to the programmer. POP-C++ also has special method invocation semantics, but syntatically the method invocation statements do not differ between local and remote invocations. Furthermore, parallel objects are shareable, that is, references to an object can be passed in any method, be it local or remote.

The POP-C++ runtime architecture consists of three actual objects for each parallel class the user implements: the Interface, the Broker and

the actual Object. The Interface is an object itself, instantiated in the caller side; it shares the method interface of the actual Object, giving the transparency of interaction for the application.

The Broker is the callee-side correspondent to the Interface, it receives method calls from the network, unpacks the data, calls the method on the actual Object and then repacks the return value and sends it back to the Interface. The actual Object is the implementation of the user, with the code that is to be distributed.

POP-C++ introduces two syntax extensions to C++ in addition to the declaration of parallel classes: Requirement descriptions and Method Semantics.

- Requirement descriptions: Using an associated object description, the developer can express resource requirements in the form of a hostname, the number of MFlops needed, the amount of memory needed and the communication bandwidth needed between itself and its interfaces.

- Method invocation semantics: The invocation semantic options are defined at compile time by the application programmer and can be classified in two types, Interface-side and Object-side:

  - Interface side semantics can be either Synchronous or Asynchronous; they control at which time the Interface-side method returns. In Synchronous mode, the caller waits until the method on the Object returns; this is analogous to traditional method invocation. Asynchronous methods return immediately, allowing the caller to continue execution.
  - Object side semantics can be either Mutex, Sequential or Concurrent. Mutex semantics guarantee no concurrency on the object, Sequencial semantics guarantee no concurrency on the particular method it is applied to, and Concurrent semantics allow full multi-threaded execution.

## 3. Extending POP-C++ into WSNs

To give the system architect an uniform model with which to program grid applications that use WSNs, we extended the POP-C++ model to WSNs. That means that not only should the programmer be able to instantiate Interfaces in sensor nodes to Objects running in other nodes, but also instantiate Interfaces to those Objects from inside the Grid. There should be no difference between "normal" Grid-to-Grid remote method calls and those performed from the Grid to the WSN.

```
parclass SensorNode
{
    public:
        SensorNode(int node, string machine) @{ od.url(machine);};

        async seq void setLEDs(char val);
        sync conc int getTemperature();

};
```

*Figure 1.* Basic SensorNode POP-C++ Class

Figure 1 illustrates the implementation and instantiation of an Object that runs on the WSN and receives function calls from the Grid. The implementation has methods to read temperature sensor values and set a value to be displayed on the LEDs of the node. Any node on the Grid may instantiate an Interface to this object and transparently call methods to it.

## 3.1 Compromises

Due to the low-resource nature of WSN nodes, the WSN implementation of the POP-C++ runtime support system had to occupy as little program and main memory as possible. This has caused us to make some compromises in the implementation:

- No interchangeable communication protocols: When in a Grid environment, it is not only interesting but necessary to support multiple, interchangeable communication protocols. When in WSNs, though, the communication protocol over the wireless link is very likely to be constant and global. Therefore, our implementation does not support multiple communication protocols at runtime.
- No dynamic resource allocation: While in Grid nodes the process of dynamic resource allocation is just a matter of downloading and executing a binary, re-writing program memory on WSN nodes is a very costly procedure, in terms of energy [Dunkels et al., 2006]. To diminish the need for re-programming, the system architect is encouraged to provide low level functions in addition to his applications' high-level routines; if a problem is found on the high-level code or a new application is to be run on the Grid, similar functionality can be attained from the aggregation of lower level calls.
- Limited Parallelism: Because of the very small amount of main memory available on sensor nodes, the amount of concurrent threads

that can run on a node is also very small. This means a limited amount of method calls, regardless of semantic, will be able to execute in real concurrency, and that the following incoming calls will have to be queued.

## 3.2 Addressing

To allow direct access to each individual sensor node, we had to extend the addressing method for POP-C++ objects. There is the possibility for POP-C++ Interfaces to be instantiated with a hostname parameter, forcing the Object to be allocated in that machine. We expanded this method to the sensor nodes, requiring two addresses:

- Address 1 - Point of contact between Grid and WSN: All WSNs must have at least one point of contact to the Grid. This point of contact must be able to communicate in both the protocol used by the Grid and the protocol used by the WSN, so it will most likely require special hardware such as the radio transceiver found in the sensor nodes. By taking this parameter we are able to instantiate a Proxy Broker in the appropriate Grid node, creating the logical bridge that forwards the method calls directed at the WSN.
- Address 2 - WSN Node: This address enables the Proxy Broker to direct the method calls to the correct sensor node. Its format is left open because different addressing methods can be used inside different WSNs.

## 3.3 The Proxy Broker

To allow the method calls to be forwarded into the WSN, a special Broker object has been created. This is a generic Broker that simply receives method calls from the Grid as if it was the Object's real broker, then forwards them to the WSN node. Once the WSN node returns from the method call, this Broker forwards the return value to the original caller. This creates the effect of transparency to the Interfaces of that Object; to them, the method calls are never leaving the Grid.

Figure 2 shows the Grid connected to the WSN through the Proxy Brokers; note that there may be more than one point of contact between the Grid and each WSN.

# 4. Evaluation of POP-C++ over WSNs

In this section we evaluate the overhead that our POP-C++ runtime system introduces by comparing two implementations of the following application: getting and setting an 8-bit value, that is to be displayed

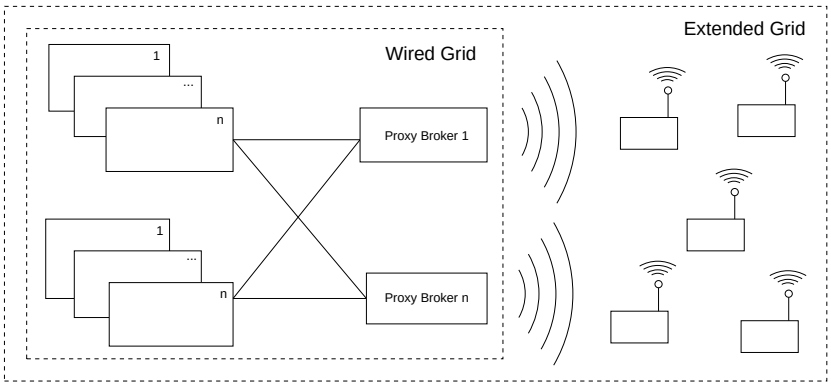


*Figure 2.* Proxy Brokers integrating the wired Grid and the WSN nodes

at the node's LEDs. One version was implemented over POP-C++ and the other directly on top of the operating system. In the case of the POP-C++ implementation, the client instantiates an Interface to a "SensorNode" Object that is executed on the other node, and calls the `get()` and `set()` methods to retrieve and set the data. In the native implementation, the client sends pre-formatted packets that are opened by the server and replied to with the data as payload.

## 4.1 Hardware testbed

The tests were made using two Mica2 sensor nodes developed at Berkeley; they use a single-channel CC1000 radio, an 8MHz Atmel Atmega128 8-bit microcontroller, 4KB of main memory and 128KB of program flash memory.

## 4.2 Runtime support

To provide the communication, memory management and concurrency support that both applications need we used the Embedded Parallel Operating System [Fröhlich and Schröder-Preikschat, 1999] (EPOS). It consists in a component-based framework for generating runtime support for dedicated computing applications. For this test EPOS' MAC protocol was configured for reliability, ensuring packet delivery through acknowledgements and minimizing network delays through an always-on duty cycle.

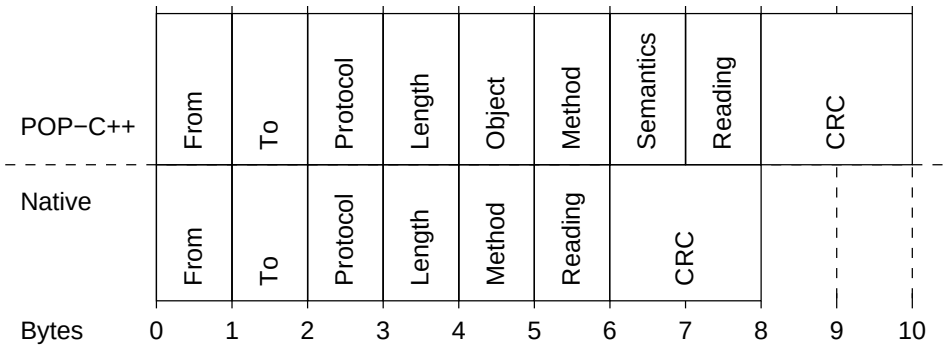


*Figure 3.* Packet Size Comparison

## 4.3 Benchmarks

- Packet size: Figure 3 details the network packet size and content for both implementations of this aplication; POP-C++ packets are larger for 2 reasons:
  - Object Field: To allow for more than one Object to be executed in each node, packets are individually addressed;
  - Semantic Value Field: The semantic values of the method to be called are also represented in the header.

  The addition of equivalent functionality on the native implementation would result in a similar packet size, which is justified by the additional information that must be transferred when supporting a complex set of applications.

- Grid-Sensor Requests-per-Second: To evaluate the overall overhead of the POP-C++ runtime system on this application we conducted a performance test that measured the Requests-per-Second that could be made from the client node to the server node. Figure 4 shows that the POP-C++ implementation could perform 6.875 remote method calls per second, and the native implementation was able to perform 7.046 requests per second. This difference of 2.42% is due to the additional processing performed by the POP-C++ runtime system as method calls arrive from the network.

# 5. Design Challenges

In this section we discuss the new issues brought by the environment POP-C++ would now work in.

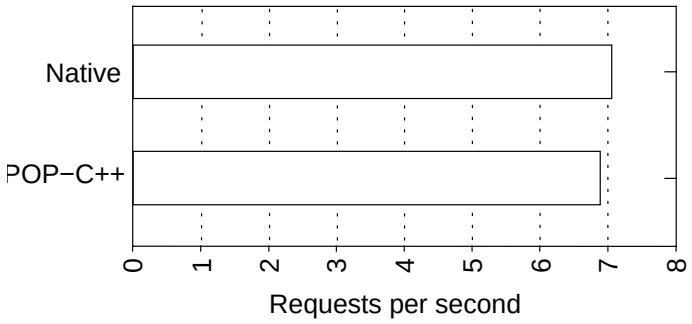


*Figure 4.* Requests-per-Second Comparison

## 5.1 Network Load

- Payload Data Overhead: To transport the additional fields necessary to make the function call, we include them in the header of our packets. This is an overhead that the original POP-C++ system also has, but it has a greater negative effect in the WSN implementation because of the high energy cost of transmitting data over the wireless link. To minimize this issue we reduced the size of each of these fields from the original 32 bit values, expecting sensor nodes to host a smaller amount of objects and methods.

- Packet Exchange Overhead: While the original POP-C++ system used additional ACKs, in the WSN implementation we leave any error correction or flow control up to the network stack of the operating system; this way, the medium access and routing protocols can handle communication in the way they see fit.

## 5.2 Scalability

If all method calls from the Grid to the WSN are routed through a single Grid node, failure of that node would cause all Objects running on the sensor network to become incommunicable. Under very heavy loads it may also become a network bottleneck, causing heavy contention at the physical level. To circumvent that, we allow multiple, concurrent points of contact between the Grid and the WSN, ideally physically separated so all can transmit concurrently.

## 5.3 Security/QoS

The issues of Security and QoS in WSNs are still relatively unexplored, but current research on these areas propose solutions at the network

level, specially at the routing layer. There are groups working on methods to protect WSNs against DoS attacks [Deng et al., 2005], guarantee packet confidentiality [Banerjee and Mukhopadhyay, 2006] and provide QoS [Ouferhat and Mellouck, 2006] on a shared WSN. POP-C++ uses the network stack of its underlying operating system as an application would, and in a similar way to the Grid implementation, we rely on EPOS to provide this kind of functionality.

## 6. Related Work

In this section we briefly discuss other projects of note that share our goal of allowing the Grid to communicate with WSNs, and discuss how their approach relates to ours.

TinyDB [Madden et al., 2003], Cougar [Yao and Gehrke, 2002] and other research efforts [Madden et al., 2002] [Bonnet et al., 2001] implement distributed query processors, putting great effort into query optimization and efficient routing. Using these techniques they have achieved considerable reduction in power consumption in addition to externalizing a more friendly SQL-like interface to the application programmer. Our extension of POP-C++ does share all these goals, but instead of active optimization, it gets out of the way of the application programmer allowing full access to the sensor nodes' hardware. Also, we see the possibility of implementing the query optimization features of TinyDB and Cougar as POP-C++ object code, yielding similar functionality.

Hourglass [Gaynor et al., 2004] inserts a Data-Collection Network (DCN) between the application and the sensor networks they acquire data from. Hourglass abstracts the internals of the sensor networks completely, and provides traditional functionality such as service registration and discovery, as well as routing the data from the sensor networks to the applications. Hourglass' approach is very internet-oriented, and uses several estabilished standards such as XML, SOAP and OGSA. In this scheme, our POP-C++ extension could be used behind the Sensor Entry Point to perform all communication in the WSNs and provide a data stream to the DCN.

## 7. Conclusion

In this article we describe a way to use remote parallel objects to integrate the Grid and WSNs, by extending the POP-C++ runtime system into the sensor network. We believe that by using POP-C++ to perform this integration we enable the application programmer to

use the WSN for multiple applications transparently, by using locally instantiated interfaces to objects that run on the sensor nodes.

When comparing a functionally equivalent application implemented with and without POP-C++, our runtime system showed a small overhead cost that was justified by the ability to support multiple applications concurrently.

## References


Banerjee, S. and Mukhopadhyay, D. (2006). Symmetric key based authenticated querying in wireless sensor networks. In *InterSense '06: Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, page 22, New York, NY, USA. ACM Press.

Bonnet, P., Gehrke, J., and Seshadri, P. (2001). Towards sensor database systems. In *MDM '01: Proceedings of the Second International Conference on Mobile Data Management*, pages 3–14, London, UK. Springer-Verlag.

Deng, J., Han, R., and Mishra, S. (2005). Defending against path-based dos attacks in wireless sensor networks. In *SASN '05: Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, pages 89–96, New York, NY, USA. ACM Press.

Dunkels, A., Finne, N., Eriksson, J., and Voigt, T. (2006). Run-time dynamic linking for reprogramming wireless sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 15–28, New York, NY, USA. ACM Press.

Fröhlich, A. A. and Schröder-Preikschat, W. (1999). EPOS: an Object-Oriented Operating System. In *2nd ECOOP Workshop on Object-Orientation and Operating Systems*, volume CSR-99-04 of *Chemnitzer Informatik-Berichte*, pages 38–43, Lisbon, Portugal.

Gaynor, M., Moulton, S. L., Welsh, M., LaCombe, E., Rowan, A., and Wynne, J. (2004). Integrating wireless sensor networks with the grid. *IEEE Internet Computing*, 8(4):32–39.

Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. (2002). Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146.

Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. (2003). The design of an acquisitional query processor for sensor networks. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 491–502, New York, NY, USA. ACM Press.

Nguyen, T.-A. and Kuonen, P. (2007). Programming the grid with pop-c++. In *Future Generation Computer Systems (FGCS)*, volume 23. N.H. Elsevier.

Ouferhat, N. and Mellouck, A. (2006). Qos dynamic routing for wireless sensor networks. In *Q2SWinet '06: Proceedings of the 2nd ACM international workshop on Quality of service & security for wireless and mobile networks*, pages 45–50, New York, NY, USA. ACM Press.

Yao, Y. and Gehrke, J. (2002). The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18.