Demo Abstract: Programming Cyber-Physical Systems with MacroLab

Tamim I. Sookoor, Timothy W. Hnat, and Kamin Whitehouse
Department of Computer Science, University of Virginia
Charlottesville, VA, USA
{sookoor,hnat,whitehouse}@cs.virginia.edu

ABSTRACT

We demonstrate *MacroLab*, which is a macroprogramming framework that offers a vector programming abstraction similar to Matlab for cyber-physical systems (CPSs). The user writes a single program for an entire network using Matlab-like operations such as `addition`, `find`, and `max`. The framework executes these operations across the network in a distributed fashion, a centralized fashion, or something between the two – whichever is most efficient for the target deployment. We call this approach *deployment-specific code decomposition* (DSCD). The MacroLab programming framework will facilitate the easy development of applications for CPSs by domain experts such as scientists and engineers with almost no additional overhead to the nodes in terms of message cost, power consumption, memory footprint, or CPU cycles over TinyOS programs.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; D.1.3 [**Programming Techniques**]: Concurrent Programming–Distributed programming

General Terms

Design, Languages, Performance

Keywords

Cyber-Physical Systems, Embedded Networks, Macroprogramming, Programming Abstractions

1. INTRODUCTION

A great deal of recent work investigates new programming abstractions and models for Cyber-Physical Systems (CPSs). However, node-level programming using TinyOS remains the defacto programming abstraction for these systems. The difficulty of software development for CPSs limits their widespread adoption. We will demonstrate MacroLab [2], a macroprogramming framework that offers a vector programming abstraction similar to Matlab for Cyber-Physical Systems.

An overview of the MacroLab architecture is shown in Figure 1. MacroLab consists of three primary components: (1) a decomposer that generates multiple decompositions of the macroprogram, (2) a cost analyzer which calculates the cost of each decomposition with respect to the cost profile of the target deployment, and (3) a run-time system which converts the decomposition into a binary executable and executes it. By decoupling the high-level programming abstraction from the low-level instantiation and generation of sensor node programs, MacroLab allows the user to write deployment-independent logic. This enables code to run efficiently across a wider variety of deployment scenarios improving portability, increasing code reuse, and decreasing overall development cost.

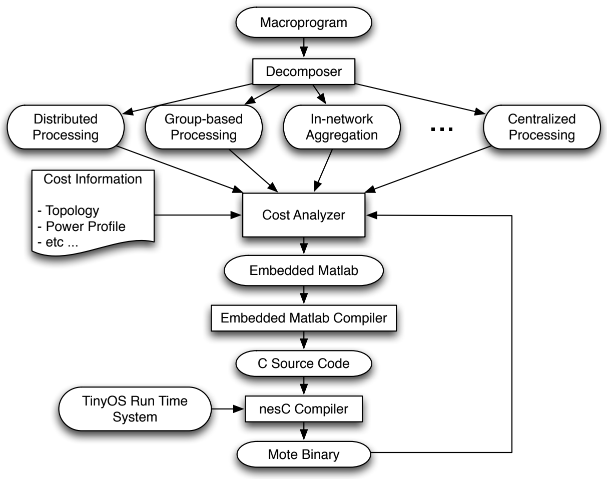

Figure 1. Overview of the MacroLab architecture. MacroLab consists of a decomposer, a cost analyzer, and a run-time system.

MacroLab introduces a new data structure called a *macrovector* which can be used to store in-network data such as sensor readings. Conceptually, each element of a macrovector corresponds to a different node in the network, but macrovectors can be stored in different ways. Each element can be on its corresponding node, all elements can be on a central server, or all elements can be replicated on all nodes. No matter how a macrovector is stored, it can support standard operations such as `addition`, `find`, and `max`. These operations may run in parallel on a distributed macrovector, sequentially on a centralized macrovector, or somewhere in between. Thus, by changing the representation

```
RTS = RunTimeSystem();
magSensors = SensorVector('MagSensor',
  'uint16');
magVals = Macrovector('uint16');
THRESH = uint8(3);
nRefl = neighborReflection(magVals,magVals>
    THRESH)
every(1000)
  magVals =  sense(magSensors);
  nodes = find(nRefl > THRESH);
  rowSum = sum(~isnan(nRefl(nodes,:)),2)
  toCheck = find(rowSum>2);
  if(toCheck)
    maxID = find(magVals(nodes(toCheck)) ...
                == max(nRefl(nodes,:),[],2));
  end
  leaderID = nodes(toCheck(maxID));
  if(leaderID)
    CAMERAFOCUS(leaderID);
  end
end
```

Figure 2. A tracking application (PEG) in MacroLab. `CAMERAFOCUS` is implemented within the RTS and sends a message to the camera, which is at the base station.

of each macrovector, MacroLab can decompose a macroprogram in the way that is most efficient for a particular deployment.

Macroprogramming environments such as TinyDB [3], Regiment [4], Marionette [5], and Kairos [1] offer a range of programming models at different levels of abstraction. TinyDB allows the user to specify the desired data using declarative SQL-like quaries while Regiment allows users to specify global operations in terms of stream operators. MacroLab is perhaps most similar to imperative macroprogramming abstractions like Marionette and Kairos. These systems support general-purpose programming with a traditional imperative programming model. MacroLab is the first macroprogramming framework for CPSs to provide vector programming, a powerful and concise abstraction that already has wide adoption among scientists and engineers.

2. MACROLAB

MacroLab allows the user to write a single program that is simple, robust, and manageable and then automatically decomposes it depending on the target deployment. Figure 2 shows the Pursuer-Evader Game (PEG) written in MacroLab. As the code listing shows, this relatively complicated application can be written in as few as 20 lines of code. A *program decomposition* is a specification of where data is stored in the network and how messages are passed and computations are performed in order to execute the program. A macroprogram may be decomposed into *distributed* operations for a large mesh network, where data is stored on every node and network operations are performed in-network. It could also be decomposed into *centralized* operations for a small star topology, where all data is collected to a central base station. A program may also be decomposed into many points on the spectrum between purely centralized or purely distributed code. The implementation could also use group-based data processing or in-network aggregation.

The MacroLab decomposer converts a macroprogram into a set of microprograms that can be executed on nodes. The goal is to preserve the semantics of the macroprogram while allowing for efficient distributed operations. The decomposition algorithm has two steps. First, it chooses a data *representation* for each macrovector, which can be *distributed*, *centralized*, or *reflected*. Based on the representation chosen, it then uses rule-based translation to convert the vector operations in the macroprogram into network operations.

In order to support the MacroLab programming abstraction, a Run-Time System (RTS) has been developed to interface commands generated by the macro-code and the low-level functionality provided by TinyOS. By restricting the generated code to only call functions within this RTS, we hide the specific details of the node's programming from the user. This also allows for optimizations to occur without affecting the running application. The user only needs to be concerned with the data, not with how the data is obtained and transported by the network. Message passing is hidden in order to provide a single machine abstraction of the network.

3. DEMONSTRATION

In this demonstration, we provide two testbeds that are similar except for their communication topology. The testbeds will be running a very simple MacroLab application and we will show that the messaging costs differ based on the topology. This demonstration will highlight the usefulness of deployment-specific code decomposition. In addition to the two testbeds demonstrating DSCD, we will have a PC from which the user can program a testbed using MacroLab.

For a more complete description of the MacroLab framework, please see our paper, "MacroLab: A Vector-based Macroprogramming Framework for Cyber-Physical Systems" [2].

4. REFERENCES

[1] R. Gummadi, O. Gnawali, and R. Govindan. Macro-programming wireless sensor networks using kairos. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2005.

[2] T. W. Hnat, T. I. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse. Macrolab: A vector-based macroporgramming framework for cyber-physical systems. In *SenSys '08: Proceedings of the 6th international conference on Embedded networked sensor systems*, New York, NY, USA, 2008. ACM Press.

[3] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, Mar. 2005.

[4] R. Newton, G. Morrisett, and M. Welsh. The regiment macroprogramming system. *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 489–498, 2007.

[5] K. Whitehouse, G. Tolle, J. Taneja, C. Sharp, S. Kim, J. Jeong, J. Hui, P. Dutta, and D. Culler. Marionette: using rpc for interactive development and debugging of wireless embedded networks. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 416–423, New York, NY, USA, April 2006. ACM Press.