Dynamic Customization in Pervasive Computing Environments

Chitra Babu, Wenonah Jaques and D Janakiram
Distributed Object Systems Lab, Dept. of Computer Science & Engg.,
Indian Institute of Technology Madras,
Chennai - 600 036, India.
Email: {chitra, wenonah}@cs.iitm.ernet.in, djram@lotus.iitm.ernet.in
URL: http://lotus.iitm.ac.in

Abstract

Pervasive computing environments are gaining prominence due to the advancements in wireless and sensor technology. This paper discusses a new approach for designing programming frameworks specific to pervasive computing environments using a **Dyn***amic* **O***bject* **Co***mposition* **La***nguage –* **DynOCoLa***. DynOCoLa differs from the traditional Object Oriented(OO) languages in that it allows the definition of objects whose behaviour can be extended at runtime. This paper describes a representative pervasive computing environment - an automated health care system. Two of the key issues in designing this system are: enabling dynamic access control to the patient records based on the location, and modifying the behaviour of sensors depending upon the context. This paper investigates how the dynamic behavioural customization feature that is unique to DynOCoLa can be leveraged in addressing the above two issues.*

Keywords: Pervasive Computing, Smart devices, Automated Health Care System, Compositional language, State Based Filtering(SBF), Aspect, Role Based Access Control (RBAC)

1 Introduction

With the advances in technology, pervasive computing environments are increasingly becoming significant. Pervasive computing environments attempt to simplify the day–to–day tasks of an individual by allowing him to perform them with the help of handheld devices[1], such as mobile phones, PDAs etc. These tasks range from simple operations such as switching on the lights in a room and checking email to complex ones like reserving airline tickets.

Smart spaces such as a room or a car, that are augmented with the ability to intelligently interact with their occupants[2] can also be viewed as pervasive computing environments. These devices in "smart spaces" can coordinate among themselves and an underlying software infrastructure to deliver relevant information and integrated services[1]. An example of a "smart space" would be a car that adapts its interfaces and capabilities depending on its occupants, or a room equipped with a heating system having the capability to alter the room temperature depending on the occupants of the room.

Since devices, sensors and sometimes even software applications in a pervasive computing environment can be accessed by users with their handheld devices, they should be equipped with appropriate access control mechanisms. Access control to both devices and software may involve assigning roles to the different entities which are allowed access and specifying their privileges. In some situations, not only the roles played by the entities but also the context in which they perform an access is important. For example, there may be a need to restrict access to important account information in a banking scenario, depending on the user's context. If the user is accessing the information via an insecure link, he may have lower privileges than if he is doing the same task using a secure link.

The growing endeavours to simplify the tasks of an individual to the extent possible by embedding devices with "intelligence", has led the application of pervasive computing environments into the domain of health care. Automated health care systems are now being developed, which enable the proactive monitoring of a patient's health in his home environment[3, 4]. The health care system is essentially a "smart space" comprising of a number of sensors and devices which interact among themselves and intelligent software agents. The sensors monitor the physiological conditions of a person as well as the environmental conditions which would in turn ascertain the health of the individual.

The devices can be software enabled, or they may coordinate with a centralized software application within the patient's home environment, that is capable of automatically generating appropriate signals based on the health conditions of the patient(s). For example, a pressure sensor may generate an alarm if the pressure of an individual goes above a threshold, and depending upon how serious the condition of the patient is, different types of alarms may be generated. Besides monitoring the health conditions of the patient, the system can automatically update the medical records of the patient maintained by a medical database at a central location, for example a hospital.

This paper discusses the use of a Dynamic Object Composition Language [5], DynOCoLa based on the Method Driven Model(MDM)[6], for realizing the features of the automated health care system. DynOCoLa differs from the traditional OO languages in that it provides language level constructs for abstracting the changing behaviour of an object into first class entities, known as *aspects*. Further, it facilitates the runtime weaving of these aspects, based on the state of the object whose behaviour should be altered. This unique ability of the language to define objects that can exhibit different behaviour based on the context, makes it well–suited for designing frameworks specific to various pervasive computing environments, such as the automated health care system discussed in this paper.

The rest of the paper is organized as follows: section 2 describes the automated health care system and highlights the issues involved in designing software for such a system. Section 3 briefly introduces DynOCoLa and demonstrates how it handles the two main issues related to the automated health care system. Section 4 discusses the related work. Section 5 concludes and provides future research directions.

2 Automated Health Care System

An automated health care system typically consists of a number of wearable and environmental sensors which coordinate among themselves and with a centralized software application. The following paragraph describes a model of a typical health care system, that depicts the entities involved and the way they communicate with each other.

Figure 1 illustrates the model of an automated health care system. There exists a centralized medical database used for storing the records of patients. This database is hosted at a server located at a main hospital. The database can be accessed by doctors, patients, and the software application that administers the patients health, which is located on a *home server* at the patient's home environment. The patient wears a number of sensors which are required for monitoring his health condition. The sensors along with the *home server* are connected via a LAN[4]. These sensors communicate the changes in the patients physiological conditions to a software application running on the *home server*. The data from one or more sensors is then assimilated to determine the health conditions of the patient. The software, in turn, has the ability to alter the records of the patient which are stored at the central database. Further, depending on how critical the condition of the patient is, appropriate signals, in the form of an automatic email, for example, may be generated to the doctor. Under critical conditions, an emergency alarm can also be generated to alert the paramedics in addition to informing the doctor.

One of the major issues in designing software for the above scenario is the need for access control mechanisms. The automated health care system consists of a number of wearable sensors and devices coordinating with each other and a software application. The software at some point of time may have to update the medical records of the patient whose health is being monitored. Besides being updated automatically, the medical records are also accessible to doctors and the patients themselves. In such a scenario, there is a need for an appropriate access control mechanism which will regulate access to these records.

Role Based Access Control(RBAC)[7] models are widely used for this purpose. Users are grouped according to the roles they play and access rights are defined for each role. These access rights are determined by the policies of the organization. When an attempt is made to access a record of the patient, the role defined for the person or software needs to be first determined, followed by the application of the appropriate access policy for the role. Sometimes the context in which the user accesses the records[8], for example, his location may also determine the access policies[9].

Another issue relates to the automated generation of appropriate alarms based on the health of the patient. A number of sensors monitor the various physiological conditions like pressure, temperature etc., of a patient whose health is being monitored. These conditions are either solely or collectively used to determine the health of an individual. If the health condition of the patient as monitored by these sensors is found to deteriorate below a threshold, appropriate alarms should be generated by the system. In complex scenarios, along with the generation of alarms, devices responsible for regulating the health of the individual or the environmental conditions, may be triggered to adapt to the patient's deteriorating health conditions.

3 Implementation

The two key issues in the development of the automated health care system are – controlling access to patient records and enabling devices with capabilities to exhibit dynamic behaviour. The former can be solved using the RBAC model, while the latter can be addressed if the devices are modeled as objects with behavioural changes. This section briefly introduces DynOCoLa, and discusses how it can be used to effectively address the above mentioned issues.

Traditional OO languages view objects as rigid entities without the ability to extend their behaviour dynamically. DynOCoLa overcomes this problem of inadequate expressiveness as follows: An object exhibiting variable behaviour is abstracted as a *partial class*, and the part of the object where its behaviour changes is modeled as a *TypeMarker(TM) method*. The dynamic behaviour of the object is captured into first class entities termed as *aspects*. The TM type of the aspect can be defined as *partof/using* or *in*. The state of the object instantiated from the partial class, is defined by its member variables and is abstracted into a state object. At runtime, the appropriate behaviour, in the form of an aspect, is *weaved* into the object during the TM method call, based on

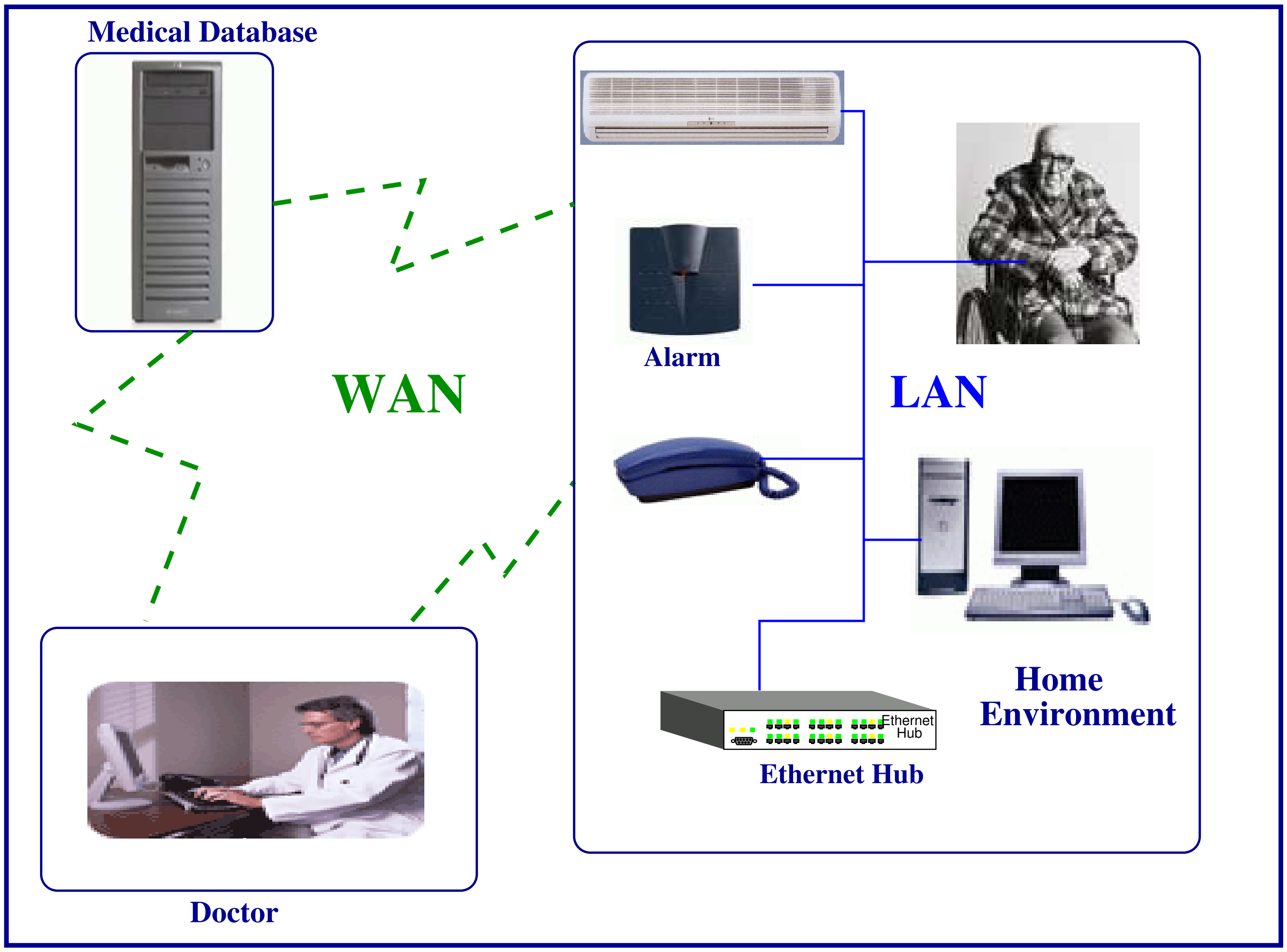

Figure 1. Automated Health Care System

the condition specified in the *unit-contract* section of the aspect. Further, since aspects define the variant behaviour, unanticipated behaviour variants can be incorporated into the application developed using DynOCoLa, at a later point of time.

3.1 Flexible Handling of Role Based Access

In the automated health care system described in the previous section, the medical records of a patient can be manipulated automatically by the software, the doctors and the patients. Consequently, it becomes extremely important to control access to these electronic patient records in a proper manner. In addition, the location from which the information is being accessed also plays a crucial role in assigning appropriate privileges to roles [10]. For example, if a doctor is trying to access the records from his office in the hospital, he should have higher privileges than if he is attempting the same task from an insecure location such as a restaurant or cafeteria outside the hospital premises.

DynOCoLa addresses this situation in an elegant fashion. The health care management adopted by a hospital, may define a number of roles, and access policies for each role, taking into consideration the location from which the access is being performed. In DynOCoLa, these roles are abstracted as aspects that define the filtering action to be performed, which in the current case is the execution of the access policy. The software system responsible for enforcing the health care policies is modeled as a partial class. The operations that can be performed on a medical record are abstracted as TM methods of the partial class. At runtime, depending on the role of the entity trying to perform an operation and the location from which the access is attempted, the appropriate aspect is weaved.

To illustrate how the language facilitates spatial RBAC, consider the following scenario: A medical record can be read or updated by users performing any one of the following four roles. These roles include those of a doctor, a patient, an automated update by a software application located in a patient's home environment, and an unauthorized user. Further, a doctor may be able to access the records from a location other than his office. In the current scenario, assume that the doctor is able to access the records only from his office at the hospital or from his home. A doctor accessing the records from his office PC through a secure LAN can read and update the medical records of any patient. However, he is not allowed to update the records from his home PC. A patient may only read his records. The software application residing at the patient's home can initiate only updates to records. An unauthorized user is not granted any kind of access.

An underlying database management system is responsible for storing the patient records. The aspect runtime is a layer that executes above this system. The access control policies are modeled by a partial class. Each of the operations, in this case, read and update, are captured as TM methods. The different variants of the TM methods are defined in the "unit-encap" section of the aspect. The "unit-contract" section of the aspect is used to specify the condition for weaving the aspect at runtime. Roles played by the entities and the location from which the access is being made, are captured as state variables. Any entity that wishes to communicate with the system has to pass its ID and any other relevant information required for authentication, along with its location, through a call to the TM method. This location is automatically sensed by the device from which the call is being made. Based on the user's ID and authentication information, his role in the system is inferred. The role information along with the location information passed as a parameter, are captured as state variables. This is done in the "plug" section of the TM method definition. The state variables in turn determine the appropriate aspect to be weaved.

It may not always be possible to know a-priori all the different locations from which an access can be made to the patient records. For example, the doctor may start using his laptop to access the records. In such a scenario, the records should not be accessible from any insecure location. Since DynOCoLa abstracts the variant behaviour of objects into aspects, it has the unique advantage of being able to handle any unanticipated change in privileges to roles, by adding new aspects and compiling them.

The technologies related to how a device senses its location are not discussed here, as they fall beyond the scope of this paper. The pseudo-code for the partial class definition and a single aspect is given in figures 2 and 3. The other aspects can be defined similarly.

3.2 Automating Health Monitoring

A number of wearable sensors and devices capture the patient's physiological information like blood pressure, heart rate etc., or environmental conditions like humidity, temperature etc. These sensors and devices in turn coordinate among each other and a software application to automatically generate appropriate signals depending on the health of the individual. There are two approaches toward the use of DynOCoLa to address the above mentioned task.

One of the possible approaches is to have the

```
public partial class PatientRecord
{
   composable−aspects:PatientRecordAspect1, PatientRecordAspect2;
   aspect−view: all;

   Hashtable recordList;   // Stores the user records

   String role;           // Role played by the current invoker of TM
   String recordId;      // record to be viewed
  String location;    // Captures the location of the  entity

  // Variables set by the aspect − if operation to be performed
  boolean canRead;
  boolean canWrite;

  Hashtable clientRoleMapping;  // Client Role Mapping

  // Constructor − can be used for variable initializations
  public PatientRecord()
  {

  }

  // TM method definitions

  TM RecordAccess
  {
      public String readRec(String client,String recId,String loc)
      {   plug
          {
              if(clientRoleMapping.containsKey(client))
              {
                 role = ((String)clientRoleMapping.get(client));
                 recordId = getRecordId(client);
               location = loc;
              }
              else
                 role = "other";
          }

          {
            // Read the record
           }
    }

   // TM method for updating the records can be written similarly

  }
}
```

Figure 2. PatientRecord.apc - Partial Class File

```
// Role of a doctor accessing records from hospital
public aspect PatientRecordAspect1   // Role of Doctor
{
  unit−type
  {
    PC PatientRecord;
    TM RecordAccess in;
  };

  unit−encap
  {
    before String readRec(String client, String recId,String loc)
    {
      thisaspect.canRead = true;
      return "";
    }
  };

  // Similarly updation filter can be declared

  unit−contract
  {
    require role.equals("Doctor") && location.equals("HospOff");
  };
}
```

Figure 3. PatientRecordAspect.as - Aspect definition for the role of 'Doctor'

aspect runtime of the language embedded in each of the devices or wearable sensors. The device or sensor is modeled as a partial class. The variable behaviour corresponding to the types of signals/alarms to be raised, is captured as a TM method. The variants of the behaviour are abstracted into aspects. The value measured by the wearable entity is captured as a state variable of the partial class. Whenever there is a change in the value of the state variable, the TM method is invoked. This invocation, in turn, causes a suitable aspect to be weaved. The weaving results in the generation of an appropriate signal/alarm.

Two of the major constraints related to sensors are low power and storage constraints[11]. Therefore, embedding the runtime of the language along with the partial class and defined aspects into the sensor itself, may not be feasible. Hence, the following approach can be adopted.

Instead of having the aspect runtime embedded into the sensors, each of the sensors can coordinate with a software application executing in the patient's home environment. This application in turn regulates the generation of signals. This application comprises of the aspect runtime, together with partial classes and aspects. Sometimes, the health conditions of a patient may be determined by the values generated from multiple sensors. In such a situation, a single partial class is used to monitor the patient's health conditions, based on these parameters from various sensors.

The following example illustrates how DynOCoLa provides a solution to the above problem. Consider a simple case of monitoring the pressure of an individual and generating appropriate alarms based on the pressure values. The pressure sensor is modeled as a partial class. The pressure values are captured as state variables of the partial class. The variable behaviour corresponding to the generation of different alarms is captured by a TM method. Four aspects are defined for various ranges of high blood pressure – mild hypertension, moderate hypertension, severe hypertension and extra severe hypertension. Whenever there is a change in the pressure value measured by the sensor, the TM method is invoked. The TM method invocation in turn causes an appropriate aspect to be weaved, based on the conditions specified in the contract section.

In very complex cases, the health of the patient may need to be monitored based on the values obtained from more than one sensor. A number of complex dependencies between various parameters may also exist, which will dictate the corresponding actions to be taken. It is impossible to foresee all the complications the patient may develop over a period of time, in such a situation.

Consider the following scenario: A patient is initially being monitored only for hypertension. However all the vital signs such as blood pressure, blood sugar level, heart rate etc., are captured as state variables. Suppose the patient develops a new complication over a period of time, such as diabetes. To handle this situation, the patient would have to be monitored also for diabetes. This should involve regulating the proper dosage of insulin based on the blood sugar level. DynOCoLa can address this newly developed complication by defining new aspects that can define appropriate actions based on both blood pressure and sugar level.

Sometimes, it may also be required to monitor the health conditions of more than one individual in the same home environment. This situation can be handled by modeling the health of each patient using a different partial class with its own set of customized aspects.

In this discussion, the availability of the technology required for the sensor to transmit the updated values to the software application, is assumed. The pseudo code for the first approach is presented in figures 4 and 5. The code illustrates the definition of the partial class and a single aspect which generates an alarm for a condition of mild hypertension. The other aspects can be defined similarly.

4 Related Work

TinyGALS[12] is a programming model designed to suit the event-driven nature of embedded systems. This employs two levels of hierarchy. At the system level, a set of modules communicate with each other asynchronously through message passing. Within each module, the components contained in them communicate via synchronous method calls.

Gay et al. have developed a dialect of C language, nesC[13] for networked embedded systems. nesC mainly focuses on event-driven execution. Applications are built using this language by writing and assembling components.

The distributed compositional language proposed in [14] discusses the various composition styles such as event and group communication and application development for sensor networks using this language.

The chief goal of above models is to achieve coordination among black-box components in embedded system applications. On the other hand, the main objective of DynOCoLa is towards dynamic customization of application behaviour in pervasive computing environments.

Lumpe et.al.,[15] have proposed a prototype composition language known as *Piccola.* This is not specific to embedded systems applications. Nevertheless, this language also deals with pure black-box components.Piccola solely addresses

```
public partial class PressureSensor
{
   composable–aspects:PressureSensorAspect1,PressureSensorAspect2;
   aspect–view: all;
   int systolic;            // Systolic pressure (90 – 240)
   int diastolic;           // Diastolic pressure (60 – 140)
   int age;

   // The partial class definition captures only pressure sensing variables
   // Same partial class may be used to monitor health based on
   // more than one parameter (Pressure)

  public PressureSensor()
  {
     // Initialize variables
  }

 // This function is called when the pressure value changes at the sensor
  public void changePressure(int sys, int dia)
  {
       this.systolic = sys;
       this.diastolic = dia;
       checkPressureCondition();    // Call to TM method
  }

  TM CheckPressure
  {
     public void checkPressureCondition()
     {

     }

  }

}
```

Figure 4. PressureSensor.apc - Partial Class File

the issue of establishing connection between the required and provided services of components. It does not deal with behavioural customization of individual components, which is the key focus of DynOCoLa.

5 Conclusions and Future Work

This paper discusses how DynOCoLa can be used to develop generic software frameworks for pervasive computing environments and automated systems. The language inherently provides this support because of its ability to model objects whose behaviour can be dictated by their state at runtime. Further, since the dynamic behaviour is captured into aspects, all the possible behavioural extensions of the object need not be known initially. They can be added in the form of aspects whenever required.

```
// Aspect defines appropriate action for Mild hypertension
public aspect PressureSensorAspect1
{
 unit−type
 {
    PC PressureSensor;
    TM RecordAccess partof;
 };

 unit−encap
 {
  replace void checkPressureCondition()
  {
   // Update medical records of patient
  }
 };

 unit−contract
 {
    require (systolic>=140&&systolic<=159)
          &&(diastolic>=90&&diastolic<=99);
 };
}
```

Figure 5. PressureSensorAspect.as - Aspect definition for mild hypertension condition

Currently, one has to code the aspects when new behaviour is to be added to an object. This process can be simplified by providing a Graphical User Interface(GUI) on top of the underlying language. The GUI will enable definitions of new aspects through the interface. This would mean that the entire customization of the system can be done using the GUI. Based on these specifications, the required aspects can be generated and compiled for use.

As future work, the language can also be enhanced to support dynamic discovery of services and devices by a mobile user as he moves from one environment to another. This feature along with the existing functionality of DynOCoLa will make the language suitable for developing applications in pervasive environments that involve mobile devices.

References

[1] Lalana Kagal, Tim Finin, and Anupam Joshi. Trust-based security in pervasive computing environments. In *IEEE Computer*, pages 154–157, December 2001.

[2] Randy H. Katz. MICRO Project #00-045: System Architecture for Smart Spaces. Technical report, University of California, Berkeley, CA.

[3] Steve Warren, Richard L. Craft, and John T. Bosma. Designing smart health care technology into the home of the future. http://www.sandia.gov/CIS/6200/Telemedicine/docs/futurehome.pdf.

[4] Ilkka Korhonen, Juha Pärkkä, and Mark Van Gils. Health Monitoring in the Home for the future. In *IEEE Engineering in Medicine and Biology Magazine*, pages 66–73, May–June 2003.

[5] Chitra Babu, Wenonah Jaques, and D. Janakiram. DynOCoLa: Dynamic Composition of Object Behaviour Through Aspects. Technical Report IITM–CSE–DOS–04–10, Indian Institute of Technology, Madras, India, 2004.

[6] Chitra Babu and D. Janakiram. Method Driven Model: A Unified Model for an Object Composition Language. *ACM SIGPLAN Notices*, 39(8):61–71, August 2004.

[7] Ravi S. Sandhu, Edward J. Coyne, Hal J. Feinstein, and Charles E. Youman. Role Based Access Control Models. In *IEEE Computer*, pages 38–47, February 1996.

[8] G. Zhang and M. Parashar. Dynamic Context Aware Access Control in Grid Applications. In *Proceedings of the 4th International Workshop on Grid Computing (GRID 03)*, 2003.

[9] F. Hansen and V. Oleshchuk. SRBAC: A Spatial Role-based Access Control Model for Mobile Systems. In *Proceedings of the Nordic Workshop on Secure IT Systems*, October 2003.

[10] Frode Hansen and Vladimir Oleshchuk. Application of role-based access control in wireless healthcare information systems. In *Proc. For Scandinavian Conference in Health Informatics*, pages 30–33, 2003.

[11] Malik Tubaishat and Sanjay Madria. Sensor Networks: An overview. In *IEEE Potentials*, volume 22, pages 20–23, April–May 2003.

[12] E. Cheong, J. Liebman, J. Liu, and F. Zhao. TinyGALS: A Programming Model for Event-driven Embedded Systems. In *Proceedings of the 18th Annual ACM Symposium on Applied Computing*, March 2003.

[13] D. Gay, P. Levis, R. V. Behren, M. Welsh, E. Brewer, and D. Culler. The nesC Lanaguage: A Holistic Approach to Networked Embedded Systems. In *Proceedings of the ACM conference on Programming Languages Design and Implementation (PLDI)*, June 2003.

[14] D. Janakiram and R. Venkateshwarlu. A Distributed Compositional Language for Wireless Sensor Networks. Technical Report IITM–CSE–DOS–04–13, Indian Institute of Technology, Madras, India, 2004.

[15] M. Lumpe, J. G. Schneider, O. Nierstrasz, and F. Achermann. Towards a Formal Composition Language. In *Proceedings of the ESEC 97 workshop on foundations of component-based systems*, pages 178–187, September 1997.