

Physicalnet: Cross-network Applications for Multi-user Sensor and Actuator Networks

Pascal A. Vicaire, John A. Stankovic
University of Virginia
pascal@cs.virginia.edu, stankovic@cs.virginia.edu

Abstract

Processors keep increasing the speed at which they can execute program instructions. The bandwidth available for both wired and wireless communications keeps growing. Specialized hardware such as sensors, actuators, and handheld devices keep improving their power efficiency and size factor. Consequently, we can expect that computers, communication infrastructures, and specialized hardware will interact more and more to form complex Cyber-physical Systems (CPSs). CPSs will revolutionize the way humans interact with the physical world. CPSs will allow the programming of applications involving globally accessible sensors and actuators, applications that are not only efficient but also robust and secure. In this essay, we focus on describing a system satisfying a possible set of CPS requirements that we motivate in the first section. As a running example, we consider an application named FireAlarm that monitors the temperature in buildings and raises the alarm in case of abnormally high temperatures.

1 System Requirements

Interoperability of Heterogeneous Devices (IH): CPSs require resource constrained sensors and actuators to interact with more powerful devices such as personal computers, cell phones, and PDAs. For instance, FireAlarm can use resource constrained MICAZs to monitor room temperature and, in case of a fire, display alarm messages on all available computer and cell phone screens.

Ease of Programming (EP): CPSs require abstractions allowing the concise programming of a large set of devices distributed over geographically separated areas. For instance, FireAlarm needs a dynamic set abstraction to specify that all the temperature sensors within a given set of buildings should sense the temperature. The dynamic set should update its membership automatically to take into account node mobility (nodes that enter/exit the buildings).

Cross-network Applications (CA): CPSs require the interoperability of devices that are part of independently deployed, geographically separated networks. For instance, it should be possible to deploy FireAlarm over all the buildings of a university, even though different administrators have independently deployed sensors in each building, with different goals in mind.

Application Concurrency (AC): CPSs require that several applications be able to use the same sensors and actu-

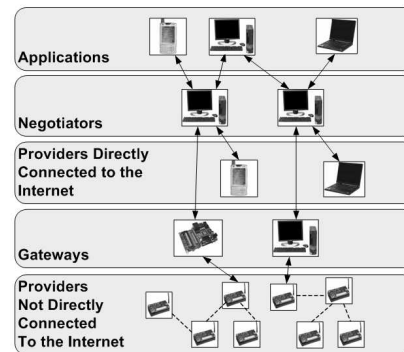


Figure 1. Architecture.

ators simultaneously. For instance, the FireAlarm application, a second application regulating building temperature, and a third application archiving temperature data should all be able to get temperature readings simultaneously.

Access Rights (AR): Because of CPS global accessibility, the CPS node owners must be able to specify user access rights to their nodes. For instance, the owner of a computer screen should be able to specify that FireAlarm can use the screen to display emergency messages, but that no other application can use the screen.

Some related work seeks to satisfy part of the aforementioned requirements but the proposed solutions cannot usually be easily extended to satisfy additional CPS requirements. For instance, the Abstract Region abstraction [1] cannot easily be extended to allow the creation of cross-network regions. Similarly, Melete [2], which supports the execution of concurrent applications on individual sensor nodes, cannot be easily extended to incorporate user access rights. Physicalnet, our solution to simultaneously satisfy the aforementioned set of requirements, is described in the following.

2 Architecture

The Physicalnet architecture is briefly summarized in Figure 1. We now explain how the various architectural elements address the mentioned CPS requirements.

IH: Physicalnet uses a service oriented architecture (SOA) that makes the services of an heterogeneous set of devices interoperable. Service providers communicate with a negotiator that maintains the list of its services and that makes these services available to application programmers.

Providers communicate with their negotiator either using the Internet or through a gateway connected to the Internet.

EP: For ease of programming, negotiators store contextual information (e.g., node location, zone definition) which applications can retrieve. Also, the Physicalnet API provides a bundle abstraction. Bundles are dynamic logical sets of services. As an example FireAlarm can define, using a logical predicate, the bundle of all the temperature sensor services that have half of their energy remaining and order all these sensors to sample the environment. As bundles are dynamic sets, sensors that do not have sufficient energy will automatically cease to be members of the bundle and will consequently stop sensing.

CA: Programmers can connect to multiple negotiators within the same application, making possible the creation of applications that use devices from multiple, independently deployed networks, as long as the network providers run the Physicalnet platform specific software. For instance, FireAlarm can connect to the negotiators of all the buildings of a university and raise the alarm in all the buildings if a fire is detected in any of them.

AC: With Physicalnet, applications can concurrently use the same sensors and actuators. Indeed, applications inform each negotiator of their service requirements (e.g., the sensor must sample, the light should be on). The negotiator memorize the list of all requirements from all applications and decides which requirement should be satisfied according to an access right table.

AR: Provider owners can specify an access right table for each of their provider services. This table is stored on the provider negotiator and determines which users can invoke which methods of which services

3 Application Example

An example implementation of FireAlarm using the Physicalnet Java API is provided in Listing 1. This particular implementation of FireAlarm displays a “Fire Alert” message on all the screens of a building, if a least one sensor within that building detects temperature exceeding a specified threshold.

From lines 1 to 5, FireAlarm creates a root bundle of all the services listed on negotiator 1 and negotiator 2. This bundle is updated every 1 second. The access rights of the application are determined by the specified user name. For FireAlarm to run on additional geographic areas, its root bundle only needs to connect to the corresponding negotiators.

From lines 7 to 9, FireAlarm retrieves the building descriptions from negotiators and specifies that the subsequent code must be executed for each building.

From lines 11 to 13, FireAlarm creates a bundle of all the temperature sensors in a building. Lines 15 to 22 specify that the bundle members must sense the temperature and be marked as “triggered” when they detect a high temperature. The bundle membership is dynamic, updated every 1 second (specified at line 2) i.e., if a sensor enters/exits the building, it starts/stops sensing the temperature. Adding 5 lines of code, we can change the bundle of temperature sensors so that it includes only those sensors that have more than one quarter of their energy remaining.

From lines 24 to 32, FireAlarm creates a bundle of all the screens in a building if one of the sensors within that

```

1  /* Bundle of all available services */
2  Bundle<Service> root=new RootBundle<Service>(1*SEC,
3  "username",
4  new String[]{"negotiator1.univ.edu",
5  "negotiator2.univ.edu"});
6
7  /* For each building */
8  ZoneList buildings=root.getZones("building");
9  for(Zone building:buildings){
10
11     /* All the temperature sensors in the building */
12     final Bundle<Temp> temps=root.get(Temp.class)
13         .get(building);
14
15     temps.forEach(new Task<Temp>(){
16         public void run(final Temp t){
17             t.samplingPeriod().set(10*SEC);
18             t.sampleReady().whenFires(new Task<Double>(){
19                 public void run(Double d){
20                     if(d>TEMPERATURE.THRESHOLD){
21                         t.putData("triggered",true);
22                     }else t.removeData("triggered");}}});
23
24     /* Bundle of all screens in a building on fire */
25     Bundle<Screen> screens=root.get(Screen.class)
26         .get(building)
27         .get(new Rule<Screen>(){
28             public boolean run(Screen s){
29                 Bundle<Temp> copy=temps.getStatic();
30                 for(Temp t:copy){
31                     if(t.hasData("triggered")) return true;
32                     return false;}}});
33
34     screens.forEach(new Task<Screen>(){
35         public void run(Screen s){
36             s.message().set("Fire Alert!");}}});

```

Listing 1. The FireAlarm application.

building is marked as triggered (if no high temperature is detected, the bundle is empty). Lines 34 to 36 specify that the bundle members must display the message “Fire Alert”. The bundle membership is also dynamic, updated every 1 second. Adding 3 lines of code, we can change the bundle so that it includes only the screens that are within 50 meters of a triggered sensor.

4 Status and Future Work

A preliminary version of Physicalnet has been implemented for PCs and MICAzs. We used the Physicalnet Java API to program 20 different applications. In the future, we plan to do a thorough performance evaluation of Physicalnet, we plan to extend Physicalnet implementation to additional platforms such as PDAs, we plan to investigate whether Physicalnet satisfies the needs of additional application domains: medical, military, environmental and industrial. Other issues that must be addressed include real-time constraints, security, efficiency, and robustness.

References

- [1] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *Symposium on Networked Systems Design and Implementation*, pages 29–42, 2004.
- [2] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun. Supporting concurrent applications in wireless sensor networks. In *Conference On Embedded Networked Sensor Systems*, pages 139–152, Boulder, Colorado, USA, 2006. ACM Press.