

A Top-Down Approach to Writing Software for Networked Ubiquitous Systems

Urs Bischoff
Lancaster University, UK

Abstract — First implementations of ubiquitous computing systems have shown promising results for a business environment. By integrating computing resources into real physical objects we can move the execution or monitoring of business processes closer to where the actual process happens. This can reduce cost and reaction time. Despite this promise there is a lack of standardised protocols. This makes it difficult for an application developer to implement business applications on top of a given network. We argue that a top-down approach to writing applications is useful. We propose a high-level language that can specify a ubiquitous network's global behaviour. A compiler can automatically generate device-level executables from this global specification.

I. What are we talking about?

Writing software can be a challenging task. Especially when we are dealing with applications in a complex environment. The ubiquitous computing scenario provides us with such an environment. This position paper focuses on an essential part of a software design process for this environment; it is about implementing applications for an environment of ubiquitous computers.

We are interested in applications in a business environment. Nowadays IT-solutions in this environment are based around a powerful backend infrastructure that collects all necessary input data in order to execute a centralised process. The ubiquitous computing vision has changed this architecture. By embedding computers into the actual physical environment, into the physical products, we can push the computation into the network. The execution or monitoring of business processes can be done closer to where the actual process happens. This can reduce cost and reaction time.

By embedding computers and sensors into objects we can make these objects “smart”; accordingly we call them *smart objects*. Our vision is a world of these smart objects and other more powerful devices (e.g. PDAs) that can provide useful services to a business. We refer to such a network as a *smart object system*.

What do we need to make a deployment of such a network feasible? We need a network that can execute the required services. If an application developer wants to deploy services in the network, then they should not need the expert knowledge of the system designer.

II. What are the problems?

The network we are addressing is highly distributed. It mainly consists of embedded computers (i.e. smart objects). Other devices (e.g. PDAs, PCs or RFID-tags) can interact with this system; they can use it or provide

services for the system. The resulting network is very heterogeneous.

Compared to traditional distributed systems (e.g. [1]) we face slightly different problems. Efficiency, for example, is defined in different terms. Throughput and latency do not play the major role in smart object systems. Because the wireless devices are normally battery powered, minimum energy consumption is an important design criterion. Dealing with this kind of problems requires a lot of low-level knowledge. This can be a challenging task for an application designer.

Because of the young nature of the field, there are not any well-established protocols for communication and collaboration between devices. It is difficult to define common protocols, because we are dealing with a variety of totally different hardware platforms. They range from passive RFID-tags to high-end PCs. It is hard to establish a “narrow waist”, or a common layer, that all devices can implement. There is no “TCP/IP” for smart object systems.

In a smart object system the network as a whole is in the centre of concern. The user is interested in the results of a running service. Knowing the individual node(s) that execute(s) this service is of less interest. This is in contrast to a more traditional point of view that focuses on a single device in a distributed environment. The ubiquitous computing vision of a large number of “invisible” devices strengthens the network-centric or global view in contrast to the node-centric one. Evaluation (at all stages) is difficult. We are dealing with new technology that allows for fundamentally new applications. Domain experts of the business environment may be unable to make use of this new potential. It is important to find a way to close the gap between domain knowledge and the way new technology can make use of this knowledge.

There is a lack of usable and stable programming abstractions or middleware solutions. Experts are needed to design the whole system (hardware and software) from scratch. It can be challenging for an application developer who is an expert in the actual business environment to implement or adapt applications.

III. The Top-Down Approach

Communication protocols and applications are traditionally organised in different layers [2]. This encourages the design of device-based software on top of established layers. This approach works well in a traditional network where we can find these well-established layers (e.g. TCP/IP). We described the

problems of establishing common layers in a smart object system. Furthermore, we showed that the network as a whole as opposed to single networked devices is in the centre of concern. In contrast to this bottom-up approach we favour a top-down approach.

We propose an abstraction that allows an application developer to write network services in a high-level language. This definition of a service does not have to specify where and how this service is executed in the network; it only specifies what the network as whole has to do and what results the user expects.

Related projects identified that many business processes are described in form of rules [3]. Rules are very natural - they can be understood by both humans and computers. We therefore developed a rule-based language for smart object systems.

In the following we show a simple example. For illustration purposes we have simplified the language. In this scenario we want to implement a service that can detect when a storage room is too hot for products that need to be chilled.

```
SPACE(storage), TIME(SIMULTANEOUS) {
    STATE tooWarm :- product(X),
                    hasToBeChilled(X),
                    hot().
}
```

The network is in a state *tooWarm* if there is a product *X* that needs to be chilled and the storage room is hot. This rule has a spatial and a temporal constraint. The rule is only valid in a certain region and at a certain time. In our example, the rule is restricted to a region called *storage*. A rule consists of a goal (*tooWarm*) and several conditions (e.g. *product(X)*). The temporal constraint in our example specifies that all conditions have to be valid simultaneously in order to satisfy the whole rule.

An application developer might have to specify several rules. In the example above there is a condition called *hot()*. However, it does not say how warm a room has to be in order to be hot. A rule specifying that it is hot if the temperature is more than 25 degrees centigrade has to be defined:

```
SPACE(storage), TIME(SIMULTANEOUS) {
    hot() :- temperature(X),
            X>25.
}
```

Similar rules are defined for *product(X)* and *hasToBeChilled(X)*.

Each device in the network exposes an interface; it specifies the device's capabilities and properties. A device that can measure the room temperature, for example, defines a capability *temperature(X)* in its interface. Other devices have different capabilities. In order to provide the whole service, they have to collaborate.

By using the temporal and spatial constraints we can define the global behaviour of a network. The declarative nature of this language allows us to separate the definition from the execution of a service. Another

advantage of this declarative language is its implicit parallelism; rule conditions that could be executed in parallel can be easily extracted.

Analogously to a compiler for a single device application we can have a process that can translate the rule-based service description into a distributed application for the given network. The network is given by all the interfaces of the devices in the network.

The service description does not specify where and how a service has to be executed; it could be centralised or fully distributed. The choice is left to the translation process. This process has to decide which nodes require what knowledge and how they can collaborate in order to provide the specified service. The translation can be influenced by a translation policy; minimising overall energy consumption is one example of such a policy. This translation process is the biggest challenge of this top-down approach.

IV. Conclusion

In order to make ubiquitous networks more accessible to application developers we need programming abstractions. It is important to find a trade-off between a problem specific language and a very generic abstraction. We proposed a rule-based language. Rules are widely used to specify business processes; the translation of such a natural language rule into an application is straightforward. Furthermore, the domain expert is familiar with rules. By hiding the complexity of the underlying network the application developer can focus on the correctness of the service rules. Evaluation time can be reduced.

The proposed top-down approach is suitable for heterogeneous networks. This approach does not require all nodes to have the same common communication protocols. We still require them to be able to communicate with each other. However, we do not build applications on top of certain common layers.

By automating the translation process the application developer does not have to deal with communication, synchronisation or other low-level optimisation problems which make distributed application complex and error-prone.

References

- [1] The Message Passing Interface (MPI) Standard. Available: <http://www-unix.mcs.anl.gov/mpi/>
- [2] Open Systems Interconnection – Basic Reference Model: The Basic Model. ISO/IEC 7498-1, 1994.
- [3] M. Strohbach, H.W. Gellersen, G. Kortuem and C. Kray. Assessing Real World Situations with Embedded Technology. In *Proceedings of the Sixth International Conference on Ubiquitous Computing (UbiComp)*, 2004.