A distributed shared data space for personal health systems

Wim Stut[a,1], Frank Wartena[a] and Maarten van Steen[b]
[a] *Philips Research, Eindhoven, Netherlands*
[b] *Vrije Universiteit, Amsterdam, Netherlands*

Abstract. Ubiquitous computing is a promising paradigm to support health care outside traditional care institutes. Sensor-based systems may continuously collect data on a person's health status and context, and provide immediate feedback or contact a remote physician. This paper presents a novel programming model to facilitate the development of such systems. The model, which has been inspired by tuple spaces, offers robustness for ad hoc mobile environments and explicit support for data streams
Keywords: Medical Informatics, Ubiquitous Computing, Sensor Networks, Body Area Networks, Intermittent Connectivity, Middleware.

1. Introduction

Demographic developments in society lead to an increasing demand for health care. The main reasons are that people get older and become increasingly aware of disease risks and a healthy life style. At the same time health-care budgets are under pressure. These two developments demand for new ways of delivering health care services.

One approach is to support people in having a healthy life style or managing their disease while residing outside traditional health care institutes. In this case, sensor-based systems continuously acquire data on a person's health status and context. These data can be interpreted locally by the system, which may give immediate feedback or control an actuator, or be sent to a remote care provider for further interpretation.

For example, when a patient is discharged from a hospital after heart surgery, a sensor may continuously measure the patient's heart rate. If the heart rate value exceeds a threshold, the patient's cellular phone automatically warns a remote physician. This concept may also be used to coach a person training for a marathon: while running, a small wearable computer advises the runner to adapt his or her speed based on the measured heart rate.

A system that supports an individual's health is called a personal health system. It consists of sensors, actuators, and appliances (such as cellular phones and PDAs) within the range of an individual person. A personal health system may be connected to other systems, such as hospital information systems, tele-health-care services, or fitness centres. An important requirement is that it should be able to support persons with multiple diseases, and be extendable with new hardware and software.

[1] Corresponding Author: Wim Stut, Philips Research, High Tech Campus 31, 5656 AE Eindhoven, Netherlands. E-mail: wim.stut@philips.com

As persons move around, devices or networks may become unreachable. Hence, personal health systems are in fact ad hoc mobile computing environments that show intermittent connectivity. Yet, robustness is a core requirement for these systems. Another characteristic of these systems is that they deal with data streams rather than individual data values as they continuously monitor a person's status and context.

There is currently very little support for building personal health systems. Applications are generally tailored to specific hardware and poorly integrated with other applications. We believe that lightweight middleware may solve many of the issues needed to develop open and portable applications that can operate independently of the characteristics of specific hardware devices such as sensors.

As a step in this direction, we present a novel programming model for the collection, exchange, management, and access of data in personal health systems. The paper is organised as follows. Section 2 describes the problem that we solved. Our programming model is explained in Section 3. Section 4 summarizes a prototype implementation. Finally, we discuss our work in Section 5.

2. Material and Methods

2.1. Problem description

Today, personal health systems are typically aiming at a single disease or purpose. This leads to unnecessary duplication of equipment and makes it difficult to offer integrated health support when patients have multiple diseases or goals. For example, if the patient and the runner from the previous section are the same person, with current technology he or she has to wear two heart rate sensors, and both a mobile phone and a wearable computer when running. Moreover, the sports application does not know that this person has had heart surgery, and cannot adapt its advice accordingly.

Furthermore, each personal health system has its own architecture, which makes it difficult to reuse components. Each has its own solutions for robustness and handling ad hoc mobility, or does not support these features at all. From a system development point of view, it would be much more efficient if these systems could be based on a shared architecture. Functionality that is needed by multiple applications can then be offered by a common middleware layer, which can support a range of devices.

As a contribution to solve these problems, we have developed a programming model for the collection, management, and access of data in personal health systems. The model is based on a shared data space concept, and explicitly offers robustness in ad-hoc mobile environments. The model hides sensors for applications: each application merely specifies what kind of data it needs (like heart rate) and at which frequency; the middleware finds the appropriate sensor, and warns the application if it cannot meet its request. The patient data as collected and stored in the data space can serve as basis for multiple applications, each targeted towards a specific disease.

2.2. Scenario

To further illustrate the problem domain, consider the following scenario:

Jim likes running. To improve his performance he wants to know his heart rate during his training sessions on a per-second basis; he therefore buys a heart rate sensor and a

PDA that communicate wirelessly. Unfortunately, several times a week Jim feels dizzy. His family doctor suspects a small cardio-vascular problem, and wants to know Jim's heart rate and blood pressure during the day on a per-minute basis. The heart rate sensor that Jim already has, is accurate enough, and can thus be used for this purpose too; for measuring blood pressure, another sensor is used. The collected sensor data are stored on Jim's PDA such that the doctor can inspect these data at Jim's next visit.

What makes this scenario interesting from a system's perspective?

- the sports application and the cardio-vascular application share the heart-rate sensor but have different sampling frequency requirements. When Jim is not running, the heart rate needs to be collected only once per minute. We could unnecessarily drain batteries when collecting these data every second.
- the cardio-vascular application needs sensor data only once per minute. Even if more data are available (due to the fact that Jim went running and the sports application asked for heart rate values every second), these additional data items need not be used by the cardio-vascular application.
- the sensor data must be available for future use in the order that they are produced by the sensor. In particular, when Jim, at his next visit, tells the doctor that he felt dizzy yesterday around 10 am, the doctor may connect her PC to Jim's body area network (BAN) to visualize the heart rate and blood pressure values as measured between 9 AM and 11 AM.
- the collection of data is orthogonal to the processing of data. At Jim's first visit to the doctor, the doctor sets the desired frequency to one sample per minute. The software that is used for this may run on the doctor's PC that is temporarily connected to Jim's BAN. However, the collecting of sensor data must continue when the associated application is not connected.
- the system's reaction to the removal of the heart rate sensor depends on the applications that are present. It would be no problem at all if, before his first visit to the doctor, Jim turns off his heart rate sensor after running. However, if this sensor is also used for collecting heart rate values for the cardio-vascular application, the system should warn Jim.
- details about the sensors should be hidden to the applications. The applications are merely interested in data that reflect the status of the person; whether the heart rate is measured by one or multiple sensors is irrelevant to the applications. Implementation changes in sensor technology should not affect the applications that use sensor data.

3. Programming Model

To address these issues, we have developed a programming model inspired by Linda-like shared data spaces [1]. Data spaces have shown to be a powerful concept when it comes to separating applications in time and space: the components do not need to co-exist in time for them to communicate and need not know about each other's existence. The asynchronous and connectionless programming paradigm of data spaces makes them more attractive for ad hoc mobile computing environments than the remote procedure call model (systems based on the latter model are less robust when devices or networks become unreachable).

In contrast to other approaches, our shared data space works on data streams rather than only single data items, as is normally the case. The components in our system (sensors, actuators, and applications) communicate via a distributed shared data space. These components do not need to know each other; instead they communicate via typed data elements. The components write and read data elements to and from the data space via generic operations.

Typically a data space contains data of a single person and is distributed over multiple nodes. These nodes communicate via a wired or wireless network, and may temporarily be disconnected. The data space system software is responsible for moving or replicating data elements between the nodes of the data space. This is hidden for the component developers; they can focus on application-level functionality.

Sensor data are typically processed in the same order as produced by the sensors. The processing may take place immediately, or later. To make the programming of applications easier, the model contains an explicit stream concept, where a stream is defined as a time-ordered collection of typed data elements (see Figure 1). No assumptions are made about the frequency: even the data elements produced by a sensor that measures the heart rate only once per day, can be viewed as a stream.

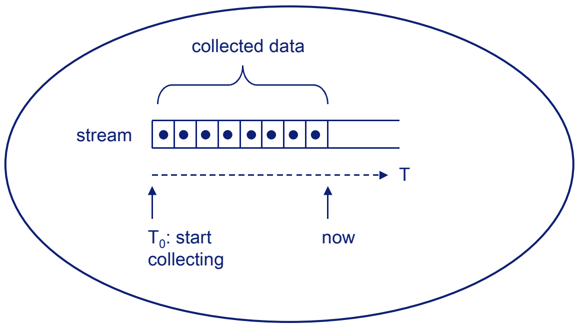

Figure 1. A stream in a distributed shared data space

Components may *create* a stream, which requires specifying the type of the data elements in the stream, such as *HeartRate*, and providing a unique name by which the stream can be identified. A stream can then be *opened* either for reading or writing elements. Since different readers may simultaneously access a stream at different positions and in different manners, opening a stream returns an application-specific *descriptor* that is subsequently used for all stream accesses by that application.

Data elements can only be appended to a stream (i.e., at the right-hand side in Figure 1), at which point they are timestamped. To read data from a stream the reader must first position itself in the stream via the *seek* operation. Data elements can then be read via the *read* operation. The effect of the read operation is that the reader gets a copy of a data element. The reader receives the data elements in the same order as they have been added to the stream.

As illustrated in the scenario, different readers of a stream may want to receive data at different frequencies. We have chosen to let the reader define its own frequency via the operation *setReadFrequency*. The side effect of the *read* operation is that the position of this reader in the stream is adapted according to its frequency. In other words, subsequent read operations return elements at the specified frequency. A read may block the caller until a sample is available.

The system may be instructed to immediately collect a data sample (which can then be read). As an alternative, an application may tell the system to start the continuous collection of data samples (at a specified frequency). We deploy call-backs to handle exceptions, such as when there is no data available.

An important observation is that the shared data space hides how it actually collects data, and from which sensors. We believe this to be an important contribution as it allows us to decouple applications from specific hardware.

4. Prototype implementation

To validate the data space concept we have built a prototype for a part of the scenario of Section 2.2 (see Figure 2). The heart rate is measured by a sensor that is connected wirelessly to a PDA via an 802.15.4 link. The PDA and laptop communicate via the network access profile of Bluetooth.

The PDA contains a sports application, which asks the data-space middleware to collect the heart rate at a frequency of 1 Hz when the user is running. The laptop (of the family doctor) contains a cardiovascular application; when Jim visits his doctor and connects his PDA to her laptop, this application asks the middleware to collect heart rate samples at a frequency of 1/60 Hz.

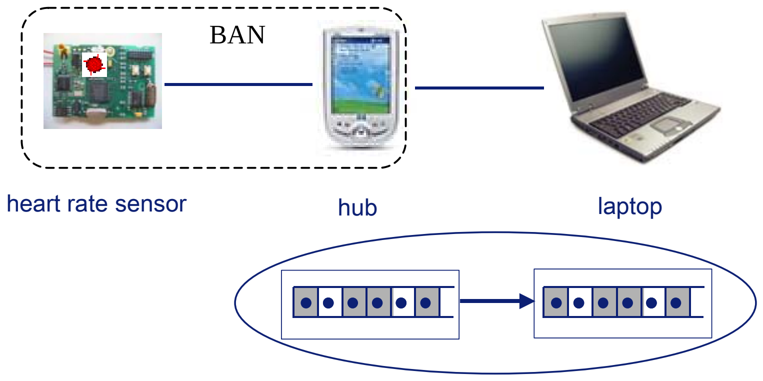

Figure 2. Overview of the prototype implementation.

Both the PDA and the laptop have data-space software. Both devices host a data-space kernel that contains stream data and that offers the data-space operations to the local applications. When the PDA and the laptop are connected, both kernels synchronize their contents by exchanging samples using an IP socket.

Once the middleware (on behalf of an application) has asked the sensor to start sampling at a certain frequency, the sensor autonomously sends the heart rate at this frequency. When a sample cannot be delivered (e.g., because the connection between the sensor and the PDA has broken), both the sensor and the PDA raise an alarm to inform the user that something is wrong. Note that this alarm is raised only after a sample could not be taken.

The prototype confirmed that the system continues to operate correctly even if a network connection breaks. The prototype did reveal that the clocks of the sensor and the PDA drift apart (about 10 ms per minute). This was an issue since the software on the sensor and the PDA need a timer to know the next sampling moment. We solved the problem by sending a clock synchronization message from the PDA to the sensor every 10 minutes to keep the drift within reasonable limits.

5. Discussion and Conclusion

This paper has presented a novel programming model to facilitate the development of personal health systems. The model, which has been inspired by tuple spaces, offers robustness for ad hoc mobile environments (data does not get lost when devices or networks become unreachable, and users are informed when the system cannot fulfil its tasks), separates applications and sensors, and explicitly supports data streams. We now briefly discuss related work for such systems.

MiLAN [2] is middleware for (medical) sensor networks. Its underlying goal is to maximize the system lifetime by reducing energy consumption. Applications explicitly specify the desired data types and quality (e.g. depending on the patient's status). MiLAN combines these requests, and determines the most feasible sensor set that satisfies the applications' requirements.

Secure UPnP [3] deals with secure access to and data transport in wireless health care systems. However, little attention is paid to data distribution with intermittent connectivity, or to the separation of applications and sensors. Fluid Computing [4] specifically addresses system robustness with intermittent connectivity, but offers no support to applications to transparently gather sensor data.

Earlier work has shown that the data space concept is a promising concept for ad-hoc mobile computing environments with intermittent connectivity. For example, LIME is aimed at applications that exhibit logical and/or physical mobility [5]. All communication takes place via transiently shared tuple spaces distributed across the mobile hosts. At any moment an application running at a host, can access the tuples located on its own host and the hosts it is connected to. The set of tuples accessible by a particular agent residing on a given host is altered transparently in response to changes in the connectivity pattern among the mobile hosts.

However, to our knowledge, existing data space models do not offer explicit support for accessing data streams and for collecting sensor data in a way that applications are shielded from low-level interfaces. We believe that by letting middleware offering this support, robust and better personal health applications can be developed. Future experiments should lead to further insight in this.

References

[1] Gelernter D. Generative communication in Linda. ACM Transactions on Programming Languages and Systems, 1985; Volume 7, Issue 1: 80–112.

[2] Heinzelman WB, Murphy AL, Carvalho HS, and Perillo MA. "Middleware to support sensor network applications". IEEE Network Magazine, 18(1):6--14, 2004.

[3] Keinänen K and Pennanen M. Secure UPnP and Networked Health Care. ERCIM News No. 63, October 2005, pp 25-26.

[4] Graf M. Fluid Computing. ERCIM News No. 54, July 2003, pp 21-22.

[5] Picco GP, Murphy AL and Roman GC. Developing Mobile Computing Applications with Lime. Proc. of the 22th International Conference on Software Engineering, ACM Press, June 2000, pp. 766-769.