

A Comparison of Two Programming Models for Pervasive Computing

Hen-I Yang, Erwin Jansen and Sumi Helal
Pervasive Computing Laboratory
Computer & Information Science & Engineering Department
University of Florida, Gainesville, FL 32611, USA
{hyang, ejansen, helal} @ cise.ufl.edu

Abstract

Establishing suitable programming models for pervasive spaces is essential in improving the productivity, enhancing the quality of pervasive systems, and creating an open platform for interoperability. Two different models, namely, the context-driven model and the service-oriented model, have been proposed and studied for their feasibilities as the foundation for implementing programmable pervasive spaces. We present these two alternatives and contrast their advantages and disadvantages.

1. Introduction

The early research projects in pervasive computing have made significant progress in exploring new applications, establishing architectures or conducting usability studies. But most of these projects required a considerable amount of customized efforts from experts for installation and diagnosis, or built as proprietary closed systems in the commercial space.

Exploring suitable programming models for pervasive spaces [1] is crucial in improving the productivity, the quality, and the interoperability. We propose and study the context-driven model and the service-oriented model, and compare their advantages and disadvantages.

SOCAM architecture [2] makes use of ontologies and predicates, and employs rule based reasoning to infer contexts. AhroD [3] is a hardware device following similar design philosophy. There are similarities to our proposed context driven model, but does not provide tightly integration between the contexts and the behaviors.

The context toolkit [4] and Gaia [5] middleware propose mechanisms to interpret, aggregate and

query contexts, and compose context-aware applications. But both focus on component brokerage rather than following through the top-down service design.

2. Two Programming Models

2.1. Context-Driven Model

A smart house is considered as a bounded environment, in the sense that there is an inside, which can be observed, and outside, which cannot be influenced. The space can be in a state that can be described using description logics [6]. There are simple states such as *cold*, *hot*, *humid*, that can be directly measured by the sensors, known as atomic contexts. More complex derived contexts are defined as combinations of atomic contexts.

At any given moment we can take a snapshot of the current observed state of the world and classify this snapshot according to the description logic.

Contexts can be ordered according to preference. For instance, "*warm and humid*" would be preferred over "*cold and dry*". When programming the smart house we would like to make sure that the eventual active context is a desirable one, or else we would like to activate available actuators that would lead to a desirable context.

Contexts that are extremely undesirable, such as "*hot and smoking*" indicating that the house is on fire, from which there is no escape, are called "*impermissible contexts*". If such a context ever becomes active, drastic emergency actions should be taken and human attentions should be invoked to resolve the situation as quickly as possible.

Following this model, system designers would first come up with the description consisting of contexts of interest for the smart house. They would then need to define the mapping between sensor readings and atomic contexts, so as to help the house make sense of these data.

¹ This is a publication of the Rehabilitation Engineering Research Center on Technology for Successful Aging, funded by the National Institute on Disability and Rehabilitation Research of the Department of Education under grant number H133E010106. The opinions contained in this publication are those of the grantee and do not necessarily reflect those of the Department of Education

A programmer also defines how actuators affect the state of the space. The effects of actuators have to be defined in terms of how they can affect the available sensors. Hence we can reason about the effect they have on the active context. Finally, for each context that is not desired we define which actuators to (de)activate.

This programming model can be formulated nicely using formal definitions by defining the operational semantics of a space and how actuators change the state of the space, providing a solid theoretical background to this novel approach.

2.2. Service Oriented Model

Service oriented model focuses on the services provided, rather than the active contexts of the smart environment. For instance, to provide climate control in the house, a service that could retrieve readings from thermometers and humidity sensors, and capable of manipulate heater, air conditioner and humidifier based on some predefined algorithm would be implemented and deployed. Services are stackable, so complex services can rely on simpler services as the provider of data or actuating entities.

In some sense, this model is a variant of the more traditional procedural programming model. But instead of focusing on defining functions or manipulating objects, each piece of software is an individual artifact that would provide some service.

3. Advantages of the Context-Driven Model

a. Explicitness. The biggest advantage of context-driven programming model is its explicitness. By describing possible contexts in a smart house using description logic, the middleware runtime and bystanders can all unambiguously identify which contexts are currently active.

By defining actions to take based on the currently active contexts, the knowledge on the behavior of system is explicit, contrasting to the encapsulated function calls of traditional programming. This is of great importance since the state of the space is meaningful and essential to its residents.

b. Interoperability and Extensibility. Entities such as sensors and actuators, both new and existing, behave based on the explicitly defined behavior in the description logic so no guessing is necessary during collaboration and integration. The explicitness greatly promotes extensibility and interoperability in an open and constantly

changing system such as a pervasive space. The separation of the knowledge definition and the availability of the entities greatly enhance its capability to adapt to changing configurations.

c. Conflict Detection. Because there is no ambiguity in identifying active contexts, and the explicitly defined associated actions to take, context-driven model is extremely powerful in detecting impermissible contexts and contradictory behaviors. Since contexts are described in the description logic, it is much easier to explicitly identify impermissible contexts.

Description logic makes testing for subsumption and membership straightforward, hence making inheritance and overriding a standard part of behavior definition. In addition, atomic contexts along the same dimension are supposed to be disjoint. All these characteristics make it much less possible to create clashing contradictory behaviors.

d. Capture Environmental Effect. Context-driven model is reactive. Instead of setting a goal and proactively trying to control and coordinate all entities to achieve that goal, systems following this model passively react to the active contexts observed by executing predefined actions associated with active contexts. The system knows what the most preferable contexts are, and where actions taken may lead to, but it remains inactive until context transitions take place. This reactive nature contributes to its capability to capture impermissible contexts and environmental effects.

e. Automatic Tuning and Machine Learning. The effect of actuators is described in terms of the transitions between available contexts. Hence it might be possible to learn the effect an actuator has on a sensor by making use of machine learning techniques, thereby rendering the explicit description of an actuator unnecessary.

f. Scalability. Unlike the service oriented model where each service is encapsulated into a stand alone software artifact, there is only one middleware bundle in the context-driven model. The bundle only takes action during context transitions, which are not expected to be frequent once the system has stabilized. The single-bundle reactive approach would fare much better in terms of scalability comparing to dozens of service bundles all proactively collecting, calculating and manipulating at the same time.

4. Advantages of the Service-Oriented Model

Contrasting the previous discussion about the context-driven model, the more traditional service-oriented programming model can be best described by as proactive, procedural and implicit. Most of its advantages and disadvantages are results of these characteristics.

a. Fine-Grained and Empowering Control. The service oriented approach offers a much more fine-grained control over the actions to take in the smart house. It takes a proactive stance when managing and interacting with the smart space to provide predefined services, rather than waiting passively for the environment to hit certain contexts and then react. The service-oriented implementation and the proactivity allows easy definition of complex stepwise controls based on the sensor readings, current status of actuators, and the past history. It is capable of providing tighter control comes because it is self-contained in defining the data source, the actuator agents, and the frequency and detailed level of control that does not rely on context transitions. For context-driven model to achieve the same level of control, a large number of contexts would have to be defined, and would probably result in hardship in human decipherable differences between various contexts.

Many objectives and interactions in the smart houses can be formulated as control problems. Some are fairly simple like stimulation and triggering, while many others can be formulated as target tracking or even coordinated distributed control. The more complex the problem is, the harder it would be to define a context graph complex enough to match the demand of these complex control problems.

b. History and Derivatives Based Decision Making. Because the whole context-driven model is built upon the concept of snapshots, one of the biggest constrain is its lack of memory; hence it is not possible to demonstrate different behaviors based on the past path leading to the current contexts. The service oriented model can follow and keep the history of the environment and past actions, and differentiate the instructions issued during the entire lifecycle of the service.

c. Stackable Services and Derived Sensors. There are difficulties in expressing some quantities that can be derived using simple mathematical

formulas when using the pure context-driven approach. For instance, it is nontrivial to introduce a context based on the mean value of readings from two thermometers without creating an off-the-graph derived sensor. On the other hand, there are multiple ways to create stacked services employing various mathematical manipulations.

d. Wide Acceptance and Proven Track Record. This can be further inspected from two perspectives. From the perspective of the programmers, since service oriented model closely resembles traditional procedural programming; it is easier to wrap their heads on the task of creating services. The major hurdles of applying context-driven model come from two sources. First, the relative novelty of context-driven model requires programmers to adapt to a new way of thinking. Second, as the domains to be considered in the smart environment increases, theoretically the complexity of identifying contexts of interest would grow exponentially. Taking full advantage of the context-driven model means not only identifying the normal contexts of interest, but all the dangerous impermissible contexts also have to be identified and accounted for. This implies that the context graph for a real smart environment would probably consist of hundreds of contexts. The task of identifying all contexts of interest and associating behaviors with each might be much more troublesome than originally thought. Programmers would more likely to embrace the more empowering and comfortable paradigm of service oriented programming.

From the business model perspective, the service provider would like to offer services that can be installed and managed remotely. The single entity viewpoint in the context-driven model gets in the way when considering the commercial prospect. The introduction of new service into a smart house programmed using context-driven model would involve examining context graphs of the target smart house, adding new contexts, and associate new action rules to each relevant contexts. The mere fact that service provider has to look into master blueprint (context graph) is enough to make both the house owner and service provider cringe. On the other hand, deploying services into smart houses employing service oriented model is analogical to installing new software on a PC. It is less labor intensive, and raises less concern about privacy and security.

5. The Best of Both Worlds

It is hard to tell which model is more suitable as the foundation of programmable pervasive computing spaces. Both models have distinct advantages, but both also have defects difficult to ignore. The context-driven model seems to be a better fit because of its explicitness, impermissible context handling and scalability, but limited expressive power and inadequacy in implementing complex applications hinders its adoption as the model of choice for pervasive computing. Therefore it is worth looking into hybrid models to see if it is possible to get the best of both worlds.

We are currently contemplating two hybrid models. The first is primarily an enhanced service oriented model with a context middleware for exception and emergency handling, as shown in Fig 1. Each software artifact provides certain predefined services, while context manager monitors and handles contradictory behaviors and impermissible contexts. A rough analogy can be drawn to the contemporary computer organization: *each piece of service software is analogous to an application installed in the computer to provide one specific service. Context management middleware would serve as the safety mechanism similar to those implemented in OS to detect memory access privilege violations, share resource conflicts, etc.* The primary concern is that unlike computers, a smart space is much more complex and open with many sharable resources. Whether this hybrid model can successfully handle such a complex system remains to be seen.

The second hybrid model, as shown in Fig 2, is an enhanced context-driven approach. With most of the original model intact, but allows composite sensors as data inlet and permits service bundles implementing complex stepwise control to serve as actuators. The enhancement greatly boosts the expressiveness and programmable control, but introduction of these entities, the floodgate is opened that knowledge is now capsulated in both the context graph and service bundles, hence compromising the crucial advantage of explicitness for the context-driven model.

6. Summary

The context-driven and the service-oriented approaches are two alternatives for implementing programmable pervasive space. The context-driven model is explicit and superb in capturing impermissible contexts and contradictory behaviors, while the service-oriented model provides a more expressive and powerful model

allowing more programmable control. Since the weaknesses of each model are hard to ignore, a possible better solution may lie somewhere in between. Two hybrid models are proposed in this paper in an attempt to get the best of both worlds.

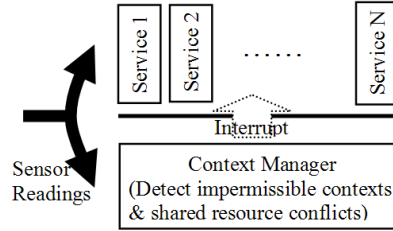


Figure 1 Hybrid Model 1

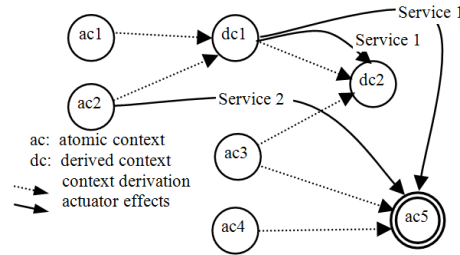


Figure 2 Hybrid Model 2

Acknowledgement We would like to thank the anonymous reviewers for their valuable comments.

References

- [1] A. Helal, "Programming Pervasive Spaces," *IEEE Pervasive Computing*, Vol 4, No 1, Jan-March 2005.
- [2] T. Gu, H. K. Pung, J. K. Yao. "Towards a Flexible Service Discovery". *Elsevier Journal of Network and Computer Applications (JNCA)*. Vol. 28, Issue 3, pp. 233-248, May 2005.
- [3] T. Terada, M. Tsukamoto, K. Hayakawa, T. Yoshihisa, Y. Kishino, S. Nishio, and A. Kashitani, "Ubiquitous Chip: a Rule-based I/O Control Device for Ubiquitous Computing", *Proc. of Int'l Conf. on Pervasive Computing*, Apr. 2004.
- [4] Dey, A.K., Salber, D., Abowd, G.D.: "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications". *Human-Computer Interaction (HCI) Journal* 16, pp. 97-166, 2001
- [5] Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt, "Gaia: A Middleware Infrastructure to Enable Active Spaces," *IEEE Pervasive Computing*, pp. 74-83, Oct-Dec 2002.
- [6] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2002.