Mobile & Pervasive Computing at the University of Florida

1. Introduction

The Atlas architecture is a framework for pervasive computing systems in the Mobile and Pervasive Computing Lab at the University of Florida, which provides a scalable network-enabled, service-oriented middleware in which numerous and heterogeneous sensors and actuators can be automatically represented as software services upon activation. It offers programmers with services and associated utilities to manage devices and compose applications without the detailed knowledge of the physical characteristics of the devices or the sensor platform node. The Atlas architecture, which includes both the middleware and the sensor platform as shown in Figure 1, provides an all-around solution to the development and deployment of pervasive computing systems.

---

2. Atlas Platform Overview

The Atlas Platform is a combination of hardware nodes and firmware running on the hardware, and a software middleware running in the network to provide services and an execution environment. Together these components allow virtually any kind of sensor, actuator, or other devices to be integrated into a common data/services bus, all of which can be queried or controlled through an interface specific to that device.

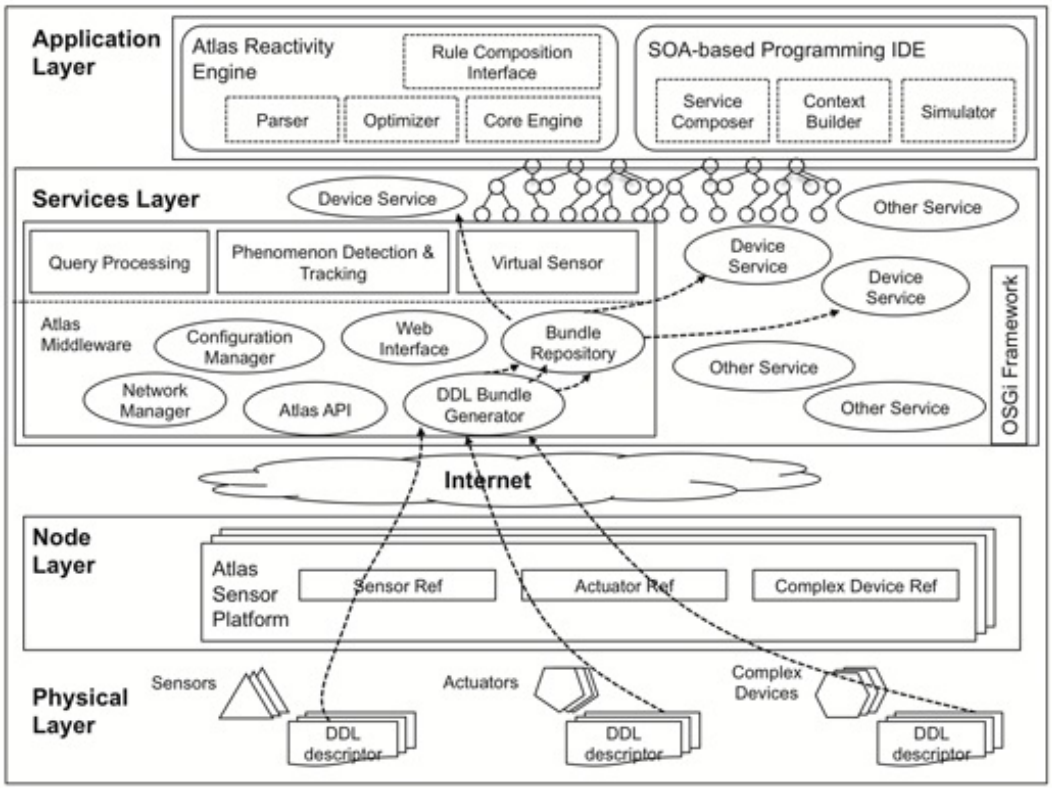


Figure 1. Overview of the Atlas reference architecture.

The hardware of the Atlas platform [Figure 2] has a modular design consisting of three stackable and swappable layers: a device connection layer that provides various connectors to devices, a processing layer that runs the firmware that controls the operation of the node, a communication layer that handles data transfer over various

networks such as wired 10BaseT Ethernet, 802.11b WiFi, ZigBee, and Bluetooth. The modular design and easy, reliable quick-connect system allow users to change node configurations on the fly. The Atlas middleware uses OSGi as its basis, which provides many service discovery and configuration mechanisms to support the creation of pervasive applications.

Figure 2. Atlas sensor platforms.

[P1] and [P2] presenst detailed introductions to the design and implementation of the Atlas hardware and middleware components, its characteristics, and several case studies of projects using Atlas.

---

3. OSGi Framework

The service layer of the Atlas architecture heavily utilizes the framework provided by OSGi. OSGi is the abbreviation of Open Service Gateway Initiative supported by OSGi Alliance, a non-profit group collaboration whose members include some of the biggest names in computing, appliance, and telecom industries, such as IBM, Sun Micro, Red Hat, Samsung, Hitachi, NEC, etc. OSGi aims to be the universal middleware. With great support in dynamic lifecycle management, this technology allows programmers to create service-oriented, component-based dynamic module system implemented in Java. It promotes high level of interoperability, and creates systems that are robust, and unflinching in dynamic environments. The specification and Java documents for OSGi R3 are avaialable at [M4] and [M5].

Current implementation of Atlas is based on Knopflerfish OSGi Service Platform Release 3, version 1.3.5. The Knopflerfish platform and its documentation are available for download at [D1] and [M3].

---

4. Deploying & Programming with Atlas

4.1 Deploying Atlas on OSGi Framework

Atlas is a two-pronged platform which consists of both hardware and middleware. In addition to deploying the Atlas hardware nodes in the distributed network, we also need to install and configure the Atlas middleware on a central server. The Atlas middlware are a set of service bundles running on the Knopflerfish OSGi service platform. These bundles provides facilties to discover, manage and interconnect a variety of services. The Atlas middleware bundles are available for download at [D2].

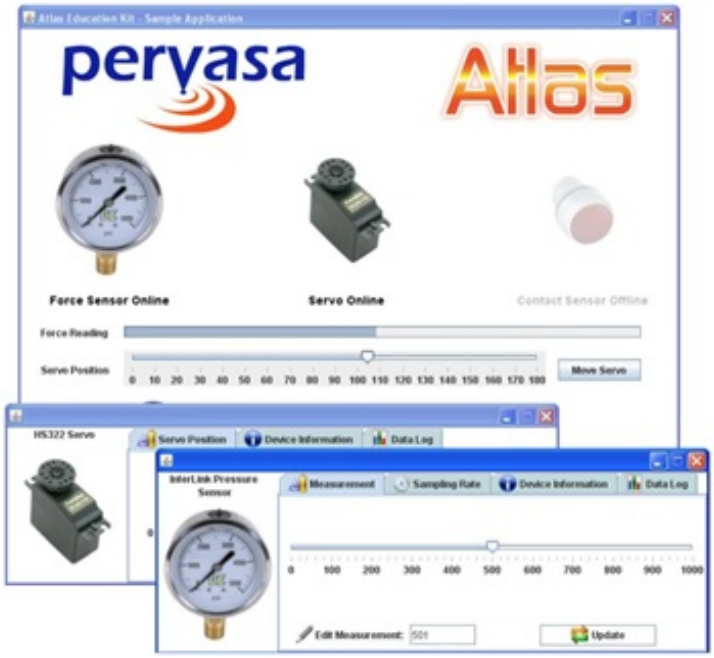
4.2 Atlas Sample Application

The Atlas sample application provides a simple example of using Atlas to develop pervasive applications. Figure 3 shows the user interface of the sample application. It monitors the status of three devices: a force (pressure) sensor, a servo and a contact sensor. Once a device is connected into the Atlas network, the application panel displays the brightend icon to indicate it is online. For sensors, the sample application receives and displays the readings on a real time basis. For the servo, it provides a slider to the user. The user drags the slider to specify a desired rotatoin angle. Accordingly, the sample application generates a command and send it to the servo over the Atlas network, asking it to rotate. The purpose of this sample application is to show programmers how to create applications in the Atlas framework, and utilize the Atlas middleware features to discover and use existing device services. The source code and documentations of the Atlas sample application are available for download at [D3].

4.3 Atlas Sensor / Acuator Emulators

The purpose of the Atlas sensor /actuator emulator is to provide a pure software environment where any sensor or actuator behavior is emulated by software. Without requiring any hardware, it provides a "hassle-free" approach to the service programmer to achieve rapid prototyping. A sensor (actuator) emulator is a single service bundle that emulates the operations of a certain type of sensor (actuator).This way, instead of serving as a proxy that passes along commands and measurements for the device, the emulator bundle has the capability to process commands and generate measurements on its own. In other words, the bundles virtually have the ability to "sense" or "actuate". In addition, the emulator provides a graphical user interface that not only displays device status such as sampling rate and measurement value, but also allows users to program these parameters to customize the device configuration.

Figure 3. Emulating devices on Atlas sample application. The above window is the sample application panel which displays the status of devices, while the bottom windows show the sensor / acuator emulator user interfaces.



The current set of devices supported by Atlas emulators include pressure sensor, digital contact sensor, humidity sensor, temperature sensor, and HS322 servo. The emulators bundle file together with a manual are available for download at [D4].

4.4 Atlas Programming API

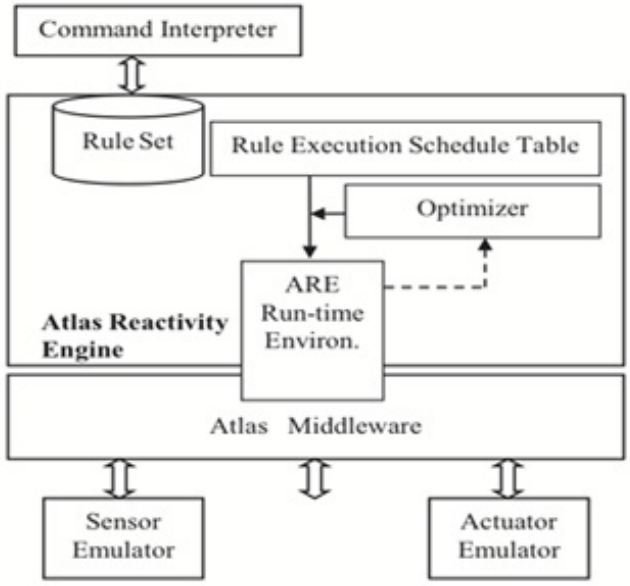
Atlas is a service-oriented platform, in which various software components and hardware devices are represented as services. Services can interoperates with each other, and composed together to form various applications. Atlas provides a universal service interfaces to developers. By extending these interfaces, the service are capable of utilizing the abundant features provided by the Atlas middleware. [M1] provides the Java documents for the Atlas service API.

---

5. Atlas Reactivity Engine

The Atlas Reactivity Engine (ARE) is an implementation of the event-driven SODA, or E-SODA programming model within the Atlas architecture. In this model, application logics are represented by a set of rules, each of which follows a typical Event Condition Action (ECA) structure. By constantly checking on sensor readings and updating rule evaluations, the pervasive system listens to certain events in the space and responds to them by taking specific prescribed actions. Compared to pure SODA, this extension model enables a streamlined and constrained way for program logic formulation that is more stringent and less error-prone. In addition, event composition creates a tighter programming space than unrestricted service composition, reducing possibilities of false, non-intentionally erroneous, or impermissible executions in the pervasive space. Furthermore, the rule-based nature of this extension model encourages a centralized reasoning engine where conflicting applications logics can be easily detected and resolved.

Figure 4. Architecture of the Atlas reactivity engine.



ARE's basic elements are rules defined over SODA services. By specifying the events, conditions and actions, programmers formulate rules that describe the desired/allowed behavior of the space in various situations. To ensure the adherence to these rules, the reactivity engine constantly checks sensor data and evaluates the rules. When certain events happen and a rule evaluates to true, corresponding actions (e.g. actuating a device or invoking another service) will be taken to respond to the event. In addition to the regular the ECA structure, ARE also provides a Time-Frequency Modifier (TFM) operator that enables a per event relaxation of rule evaluation. The relaxation is intended to be specified by the programmer based on application and event semantics. The full description of ARE rule grammar and TFM can be found in [P3]. A reference implementation of Atlas Reference Engine can be found at [D5].

---

6. Resources

6.1 Manuals and Documentations

- [M1] Atlas Developer API Java docs. (link)
- [M2] Atlas Sensor / Actuator Emulator Programmers Manual. (pdf)
- [M3] Knopflerfish OSGi 1.3.5 Documentation. (link)
- [M4] OSGi R3 Specification. (link)
- [M5] OSGi R3 Java docs. (link)

6.2 Downloads

- [D1] Knopflerfish OSGi Framework 1.3.5. (download)
- [D2] Atlas Middleware Bundles with OSGi. (download)
- [D3] Atlas Sample Application. (download)
- [D4] Atlas Emulators. (download)
- [D5] Atlas Reactivity Engine Reference Implementation. (download)

6.3 Publlications

- [P1] J. King, R. Bose, H. Yang, S. Pickles and A. Helal, "Atlas – A Service-Oriented Sensor Platform," Proceedings of the first IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006). Tampa, Florida, November 2006. (pdf)
- [P2] R. Bose, J. King, H. El-zabadani, S. Pickles, and A. Helal, "Building Plug-and-Play Smart Homes Using the Atlas Platform," Proceedings of the 4th International Conference on Smart Homes and Health Telematic (ICOST), Belfast, the Northern Islands, June 2006. (pdf)

- [P3] C. Chen, Y. Xu, K. Li and A. Helal, "Reactive Programming Optimizations in Pervasive Computing," Proceedings of the 10th Annual International Symposium on Applications and the Internet, July 19-23, Seoul, Korea. (pdf)
