appliedinformatics

Building Applications for Embedded Linux Systems with the Applied Informatics C++ Libraries and Tools

Günter Obiltschnig
Applied Informatics Software Engineering GmbH
St. Peter 33
9184 St. Jakob im Rosental
Austria
guenter.obiltschnig@appinf.com

Software for network-enabled smart devices becomes ever more sophisticated and complex. This complexity is driven by the many requirements that those devices must satisfy. For example, a built-in web server used for device configuration, administration and monitoring is considered a standard feature today. In the past, a device that needed to exchange data with a host computer typically had a serial interface and used a simple, often homegrown protocol for sending messages over that interface. In modern devices, the role of the serial interface for host communication is being taken over by an Ethernet interface. Instead of a simple binary protocol, a rich XML based protocol is commonly used. A glimpse into the near future shows us that devices will soon be required to provide and consume SOAP (Simple Object Access Protocol) based web services.

Traditionally it was common that vendors of embedded devices created the entire software stack themselves, except maybe for the operating system, if one was used at all. This is no longer feasible today. Especially if Linux is used as operating system for the device, a rich collection of free open source libraries and applications is available for developers to chose from, covering features like XML parsing, network protocols (HTTP, etc.) and so on. While these libraries can be used without paying license fees, their use is nevertheless not without cost. Licensing, legal (GPL) and support issues aside, these third-party libraries and applications must first be integrated into the development environment and the build system, and developers must learn to use them effectively and efficiently. Especially if third-party C++ libraries are used, different libraries often provide overlapping functionality. For example, most of the class libraries that have been developed before the C++ standard library became widely available come with their own string and collection/container (arrays, lists, etc.) classes. This hurts in three ways. First, scarce memory is wasted for the same functionality being unnecessarily implemented multiple times. Second, code that uses these libraries must often convert data structures from one representation to another. Third, developers have to invest additional learning effort, knowing the details and intricacies of all libraries. This gets worse if different libraries use different coding styles and design philosophies.

In 2004, Applied Informatics started with the development of the POCO C++ Libraries. The goal was to come up with a collection of portable, cross-platform C++ class libraries that can be used to create network centric applications in technical and embedded computing. Currently at version 1.3, these libraries cover a wide range of functionality, from basic operating system abstraction (filesystem access, multithreading, etc.), to XML parsing, to networking (TCP and UDP sockets, HTTP server and client, FTP, SMTP, POP3, SSL/TLS, etc.). The core libraries are developed in an open source project (http://pocoproject.org), and are freely available for both open source and commercial use, under a very liberal open source license. Built on top of these open source libraries, Applied Informatics provides

additional C++ libraries and tools. This includes support for **Fast Infoset**, a very efficient binary encoding of XML, standardized by the ITU and ISO as ITU-T Rec. X.891 and ISO/IEC 24824-1. With Fast Infoset, considerable performance improvements can be gained if XML formatted data is to be sent across a network. From the developer's perspective, it is completely transparent whether XML data is serialized as plain-text XML, or binary Fast Infoset, as the Fast Infoset library is completely integrated with the XML library. Fast Infoset is a great choice if data is to be exchanged with host applications written in Java or .NET, as Fast Infoset implementations are readily available for those platforms. The interoperability of the different implementations has been verified.

The **Netconf** library provides support for the NETCONF protocol, a standardized (RFC 4741) protocol for secure remote device configuration and management. The NETCONF protocol has been developed by the IETF NETCONF working group, with the participation of companies like Cisco, Juniper Networks, Nortel and Motorola. With the NETCONF protocol, XML-based device configuration data can be securely transferred to, or retrieved from a device in the field. Also, different configuration sets can be managed in a standardized way.

Applied Informatics **Remoting** is a comprehensive framework for building distributed applications and SOAP-based web services. Remoting can turn an existing C++ class into a network or web service. For this, a C++ class declaration has to be annotated with special comments. A code generator uses these annotations to generate the client-side and server-side marshalling, proxy and stub code. Different transport protocols are supported, e.g. a highly efficient binary transport protocol and an XML-based protocol that supports SOAP and WSDL and can be used to communicate with Java and .NET-based host applications. Figure 1 shows a C++ class declaration containing annotations for turning this object into a web service.

```
//@ serialize
struct Measurement
{
    Poco::Timestamp time;
    double          temperature;
    double          humidity;
    double          pressure;
};

//@ remote, synchronized=true
class WeatherStation
{
public:
    void getMeasurements(
        int count,
        std::vector<Measurement>& measurements);

    // ...
};
```

Figure 1. A C++ web service built with Remoting

The **Open Service Platform** (OSP) is a C++ based middleware providing a powerful plugin and services infrastructure for developing, deploying, running and managing modular network-based applications. As such, OSP is to C++ to what the OSGi™ Service Platform, developed by the OSGi Foundation (http://www.osgi.org) is to Java (OSGi is a trademark or a registered trademark of the OSGi Alliance in the United States, other countries, or both). At the core of OSP lies a powerful software component model based on the concept of bundles. A bundle is a deployable entity, consisting of both executable code and the necessary configuration, data and resource files (e.g. HTML pages) required for running the code. Bundles extend the functionality of an application by providing services to other bundles, end-user functionality or web services. A central Service Registry allows bundles to discover the services provided by other bundles. Bundles can be added, updated, started, stopped or

removed from an application without the need to terminate and restart the application. With the optional web-based administration console provided by OSP, this can even be done remotely with a web browser.

This plugin-based architecture of OSP addresses an increasing problem in software development: the large number of application configurations that need to be developed and maintained. The standardized OSP component architecture simplifies this configuration process significantly.

The Open Service Platform is based on a layered architecture, depicted in Figure 2. At the core of OSP is the *Portable Runtime Environment*, consisting of the C and C++ standard libraries and the POCO Core Libraries (Foundation, XML, Util and Net). Layered above the Portable Runtime Environment is the OSP Framework, consisting of *Service Registry*, *Life Cycle Management*, *Bundle Management* and *Standard Services*. Application-specific bundles based on the OSP Framework implement the actual application logic.

The Portable Runtime Environment sits at the center of the OSP architecture. Based on the C and C++ Standard Libraries, as well as on the POCO Core Libraries, it provides platform-independent low-level services to the upper OSP layers, such as:

- access to the file system
- multithreading support
- shared library and class loading
- notifications and events
- logging and error reporting
- XML parsing
- configuration data handling
- TCP/IP sockets and support for various network protocols (HTTP, FTP, SMTP, POP3, etc.)
- various utility classes and functions

By isolating applications from the operating system interfaces, the Portable Runtime Environment makes it possible to write applications that can be compiled for and run on different operating system platforms and processor architectures, all from the same source code. This is great for projects consisting of both embedded Linux systems and Windows or Unix-based server or client applications like building automation or industrial automation systems. The same code base can be used on all platforms, significantly reducing the development effort.

The Service Registry allows a bundle to register its services to make them available to other bundles, as well as to discover the services provided by other bundles. Since bundles can appear and disappear in an application at any time, the Service Registry also provides notification mechanisms so that a bundle can be informed when another bundles it depends on is removed from the system. This also creates the foundation for building smart devices that are capable of being upgraded with new software features, even when already deployed in the field.

OSP comes with a number of standard services implementing commonly required features. Examples include a built-in HTTP/HTTPS server, user authentication and authorization, support for sending email messages, web-based bundle management and remote monitoring, as well as a configuration and preferences storage service.

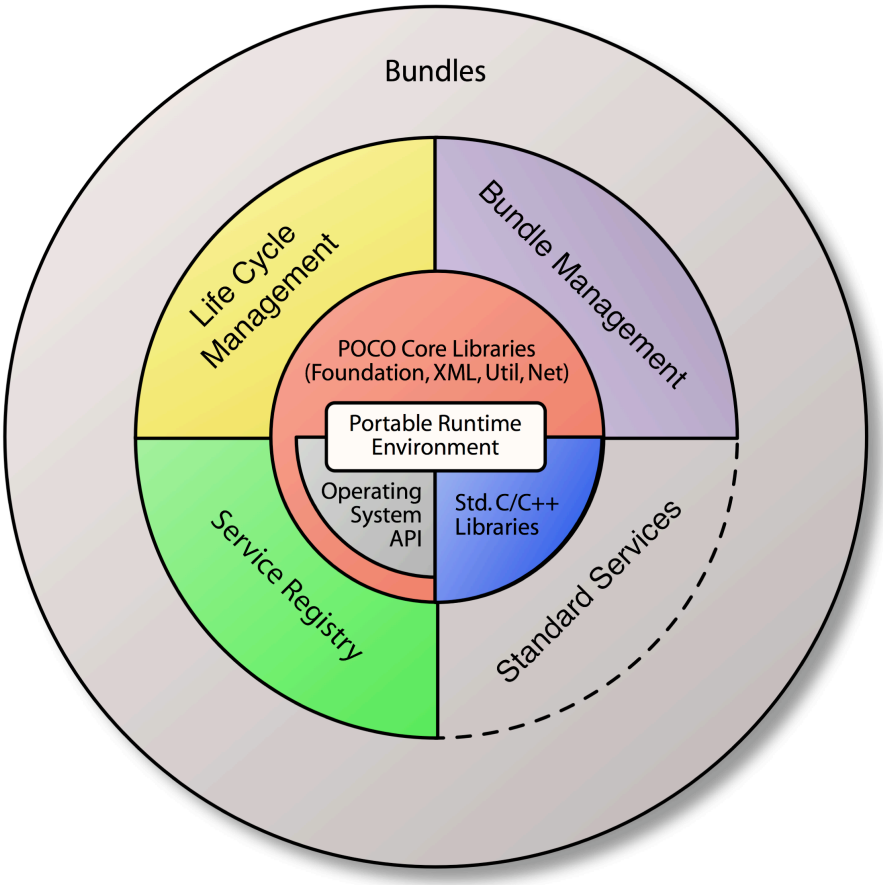

Figure 2. Open Service Platform layered architecture

All libraries and tools are implemented in highly portable ANSI C++. Both the libraries, as well as applications built with the Applied Informatics C++ libraries can be easily ported to another operating system with minimal effort, usually just by recompiling. The hardware requirements are modest. Popular ARM-9 based microcontrollers (e.g., the Atmel RM9200) with a minimum of 32 MB of RAM are a perfect target platform for deploying POCO-based applications.

With the Applied Informatics C++ Libraries and Tools, a comprehensive and fully integrated platform for the development of network-enabled smart devices based on embedded Linux is available. This platform can greatly reduce the development risks and costs and significantly improve the productivity of developers. This will lead to a faster time-to-market for new embedded Linux based products. The Applied Informatics C++ Libraries and Tools have been used with great success by customers in different industries, ranging from industrial automation, data acquisition, automotive test systems, building management and consumer electronics.