Device communities: Embedded here, there, and everywhere is now

Today, your toothbrush cannot scold you to floss more. Nor can your smart phone actively seek out new mobile social networks in real time or enable ad hoc *gaming with other people in your immediate location. It seems almost utopian to imagine a world where junk mail and spam are replaced by location-based, opt-in advertising. These ads would be proactively delivered by intelligent software that knows end users' interests and needs. A world in which autonomous, intelligent devices communicate, collaborate, and network is a concept hitherto relegated to the Sci-Fi channel. Yet with recent breakthroughs in intelligent middleware, collaborative device communities are possible today for the developers bold enough to embrace this new world vision.*

By Bob DeAnna

Embedded Computing Design touched on aspects of this networked society in the April 2007 article, "Embedded here, there, and everywhere," which clearly and accurately speaks to a pervasively connected world that many are just beginning to envision. Leading the way are developers of intelligent home automation, wearable computing, and intelligent clothing for health monitoring. Telematics is another area with tremendous potential for intelligent applications, such as automated, computer-driven collision avoidance, adaptive cruise control, lane departure systems, and other smart car features.

Isolated examples of next-generation applications are plentiful. Yet to grasp the true power of intelligent communities, applications must be realized on a large scale. The next evolution of mobile device communities will require a sophisticated communications infrastructure that allows *edge* devices – be they business or consumer, human or machine operated, wireless or wired – to participate in coordinated knowledge sharing, problem solving, and transactions that span dynamically assembled device communities, communication protocols, and multiple companies or consumers.

Extreme personalization also will characterize these applications, turning mobile and embedded devices into wireless learning machines that can be trained to know a user's behaviors and habits. Based on a user's location, time, day, work, or play, these devices or device communities will be able to deduce a user's needs and proactively deliver highly individualized content without requiring a user to Google for or even think about it.

Next-generation *killer* applications are not light years away. In fact, such applications are currently in development, which will be discussed later. First, some misconceptions about the state of intelligent software today must be identified.

Solving the size, cost, and performance conundrum

Previous barriers, such as the lack of powerful, inexpensive devices, inability to communicate over a wide range of wired/wireless networks, and interoperability, are becoming increasingly difficult to justify. On the hardware forefront, smart phones are perhaps the most ubiquitous

Expanding server-level intelligence

Considered a technology before its time, Voyager Edge from Recursion Software is an intelligent agent platform that has been commercially available since the late 1990s. Voyager Edge extends server-level intelligence (Figure 1) to a wide range of device communities, embedded and wireless operating systems, and software languages using varied distributed protocols and messaging capabilities. Device communities can be described as any logical collection of embedded or wireless devices, mobile Internet devices, ultra-mobile PCs, tablets, laptops, desktops, sensors, RFID readers, servers, and so on.

Voyager Edge enables peer-to-peer device communication as well as peer-to-group messaging/discovery and support for federated yellow pages. Recursion Software is currently adding Voyager support for Java Micro Edition (JME) Connected

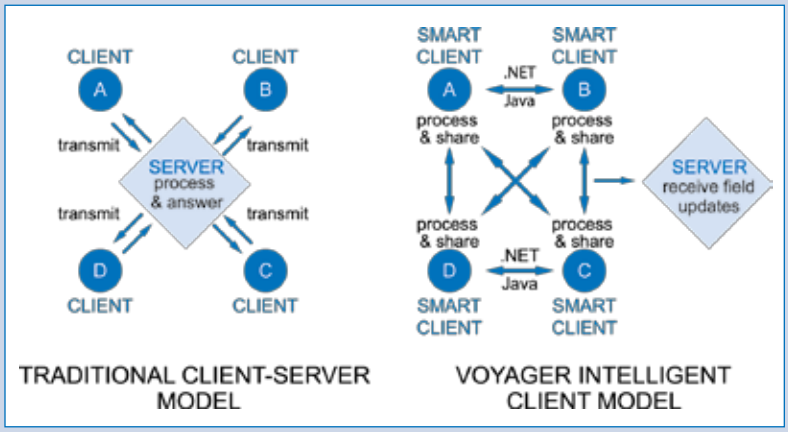

Figure 1

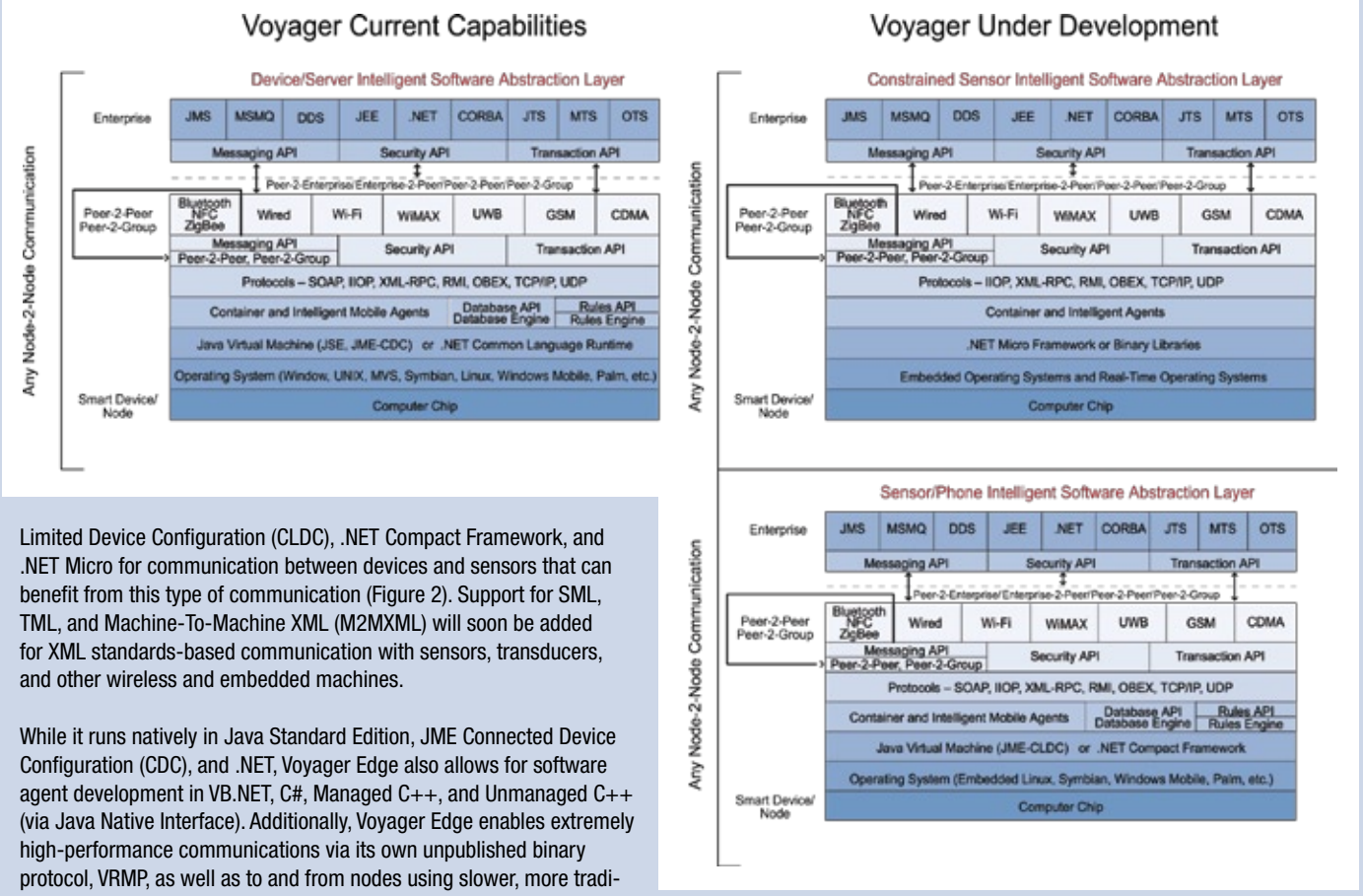

Figure 2

Limited Device Configuration (CLDC), .NET Compact Framework, and .NET Micro for communication between devices and sensors that can benefit from this type of communication (Figure 2). Support for SML, TML, and Machine-To-Machine XML (M2MXML) will soon be added for XML standards-based communication with sensors, transducers, and other wireless and embedded machines.

While it runs natively in Java Standard Edition, JME Connected Device Configuration (CDC), and .NET, Voyager Edge also allows for software agent development in VB.NET, C#, Managed C++, and Unmanaged C++ (via Java Native Interface). Additionally, Voyager Edge enables extremely high-performance communications via its own unpublished binary protocol, VRMP, as well as to and from nodes using slower, more traditional communications, such as Simple Object Access Protocol (SOAP), Internet Inter-ORB Protocol (IIOP), Remote Method Invocation (RMI), and XML Remote Procedure Call (XML-RPC).

devices, and falling prices are making even the most sophisticated handsets affordable. For embedded devices, OEMs are racing to develop increasingly powerful and smaller devices, such as Gumstix computers that are about the size of their namesake package of gum and cost about $100.

Small size comes with the trade-off of computing power. One can argue that the true hurdle to edge computing is the absence of fast, powerful, lightweight applications that can run on handhelds. Intelligent processing and transactional computing cannot occur on dumb clients where intermittent server connections, proprietary "locked" platforms, and large install footprints are prohibitive.

The inability of today's popular enterprise systems to interoperate and perform well in distributed heterogeneous collaborative environments is an obstacle that intelligent middleware can now overcome. Devices that host intelligent software components, called *agents*, can communicate to other devices directly (peer-to-peer) or to logical collections of devices (peer-to-group) in any programming language, and do so autonomously, with or without network connectivity. These devices will bridge the gaps in today's enterprise systems.

Agent platforms are not new; they have been commercially available since the late 1990s. Agent frameworks deliver superior performance over XML or Java RMI solutions because of their ability to provide server-level intelligence and complete transactions on the edge.

Additionally, microprocessor makers like Intel are providing software development kits, such as the Intel Mobile Development Kit, that allow intelligent software agents to access power, storage, wireless connectivity, memory, and processor information about the device. On an agent platform, this information can be communicated to administrators regardless of the device they are currently holding, but even more powerfully, can be acted upon to create self-managing, self-healing capabilities to the network of devices. Thus, processing is not interrupted despite network, hardware, or software failures.

Open spectrum means open phones and wireless

The high stakes battle over licensing and radio frequencies is currently being waged with the Federal Communications Commission (FCC) as referee. Google's stance on opening up the 700 MHz wireless spectrum auction is pitting the company against some of the largest telephone companies in the country. In late July, the FCC seemed to have embraced two open standards: one regarding open applications and consumers' right to download and utilize any software applications or content they desire, and another regarding open devices and consumers' right to utilize their handheld communications device with whatever wireless network they prefer. Regardless of how this issue plays out, society has never before been closer to the possibility of open devices and open airwaves.

Telcos also seem to be losing control over the networks available to consumers. Any smart phone purchased recently most likely supports three wireless networks: the traditional Global System for Mobile communications/Code Division Multiple Access (GSM/CDMA) telco network or 3G and 4G variants thereof, Wi-Fi, and Bluetooth. Some smart phones like newer Nokia handsets are starting to ship with support for Near Field Communication (NFC).

Wireless applications need a software platform that enables them to take advantage of the multiple networks available and provides the ability to communicate with peer devices or groups. These applications often must work on multiple wireless protocols, such as Wi-Fi, Ultra-Wideband (UWB), GSM, CDMA, NFC, Bluetooth, and ZigBee, as well as associated networks. The ability to dynamically reconfigure a device based on the communication protocol available to it is a very desirable capability. Processing data at the source to minimize network traffic, handling unreliable and/or limited network connections, and adjusting to hardware failures or CPU load are helpful capabilities as well.

Additional technology considerations

Interoperability: past, present, and future

Interoperability is another key issue when evaluating new technologies, as wireless systems meld with legacy wired systems and developers integrate the enterprise software systems du jour, such as Java Enterprise Edition (JEE) and .NET, with legacy software systems, such as Common Object Request Broker Architecture (CORBA), Mainframe, and so on. A lack of backward compatibility can be frustrating for companies or agencies with a great deal of legacy code or infrastructure. Even organizations with updated software stacks must choose between JEE or .NET

Collaborative device communities require real-time location-based device discovery in which a device is actively entering, leaving, and seeking other devices within the group.

architectures, thereby limiting the applications they can run regardless of their needs.

Easy installation and remote maintenance

Another common misconception is that the communication infrastructure for multiple user and device communities would be difficult to install and maintain. Again, agents solve this problem handily.

Security in a networked world

The security question is ever evolving, as are the answers to keeping application data safe. This point dovetails nicely with the previous concept that frequent software updates should occur remotely with little to no prompting from the user. However, one blanket requirement for device communities is that each device must have complete control over what services/data other devices can request of it. Furthermore, users must be able to opt in and out of services and communities easily.

Technology resistance: the only barrier is in the mind

Finally, the only remaining barriers are psychological. Some engineers remain resistant to peer-to-peer technology because client-server and hardwired software is the only model they have ever known. As with any radical directional change, it can be difficult to make the mental leap from dependence on the enterprise to intelligent clients that communicate, execute, and "live a life" apart from the server.

Other skeptics think developers already have a solution for a collaborative communication platform: the Internet. After 15 years of widespread adoption, the Internet has become the mainstay for today's mobile and distributed applications. So what's wrong with humans browsing and machines accessing Internet applications for many of the aforementioned scenarios? This presents two major problems.

First and foremost, the Internet is static in the sense that it is not truly real time and does not easily lend itself to mobility. The Internet is on a different time horizon – one could argue on a reactive timeframe – whereas mobile *ad hoc* networks bring device-to-device networking into the "now." The Internet is a passive medium that requires a user to initiate action, wait for a response, make decisions, and wait for a decision from other users or machines in return. For some applications, this limitation is perfectly fine. But for mission-critical, time-sensitive applications, for example, location-based advertising or mobile social/device networking, the moment of opportunity could easily pass by. Furthermore, collaborative device communities require real-time location-based device discovery in which a device is actively entering, leaving, and seeking other devices within the group.

Second, the Internet does not offer intelligence (yet). Applications use the Internet as a relatively easy method to send and access similar information across multiple locations, as a research tool for static content, and as the current, comparatively slow way to enable applications to interoperate (Web services). Intelligence is just beginning to enter Web search applications with semantic Web prototypes. The semantic Web uses software agents to perform intelligent searches. For example, a semantic Google search could find the lowest price of a particular pair of jeans at a store nearby a user's home address. On top of this, using a smart phone, a user could check the stock, size, and color of the jeans and have them bagged and paid for by the pickup time. The latter functionality demonstrates the all-encompassing power of intelligent device networks.

Executing on the edge

It may take some time for everyone to realize the true value of executing on the edge, especially for organizations entrenched in traditional application server and enterprise service bus solutions. But traditional enterprise is showing its age under pressure from customer demand for faster, personalized, collaborative content to the edge as well as intelligent sensor and distributed knowledge networks that can participate in complex event processing.

Though Bill Gates touted the concept of *Business @ the Speed of Thought*, it is edge and embedded devices each acting as their own intelligent servers and collaborating in a community of servers, not the enterprise, that will truly enable knowledge transactions at this speed.

The only question that remains is how fast can you code? **ECD**

Bob DeAnna *is the CTO at Recursion Software in Frisco, Texas. He has 22 years of experience in software architecture, development, and mentoring. Bob's expertise is in distributed application frameworks such as JEE, CORBA, and Application to Transaction Manager Interface (ATMI). He has architected and developed applications and middleware in Java, C/C++, Common Business-Oriented Language (COBOL), Programming Language One (PLI), and Assembler on operating systems ranging from Multiple Virtual Storage (MVS) to UNIX, Linux, and Windows. Bob received a BS in Mechanical Engineering from Rutgers University and a continuing education degree in C/C++ and UNIX Programming from New York University.*

Recursion Software
972-731-8800
bdeanna@recursionsw.com
www.recursionsw.com