Convergence Technologies for Sensor Systems in the Next Generation Networks (NGN)

Conor Gildea and Declan Barber
Institute of Technology Blanchardstown, Ireland
conor.gildea.byrne@itb.ie declan.barber@itb.ie

Abstract

This paper describes an approach to the internetworking of sensory nodes in a converged network environment. This preliminary investigation of sensory network creation is driven by a joint applied research project which seeks to establish the feasibility of the real-time remote monitoring of animal welfare while in transit between Ireland, Europe and the Middle East. This paper examines the use of Java to create sensor services in converging architectures which leverage the Internetworking protocols and describes our implementation of such a system.

Introduction

Traditional centralized static models have been applied to intercommunication between relatively unintelligent sensor nodes and intelligent management stations. Recent trends are making it increasingly feasible to move away from this centralized model to a more distributed one, even to a point where a mobile sensor network could be considered as an ad hoc network of autonomous nodes. The impact of Moore's Law has led to the concentration of greater processing power, memory and storage (and consequently increased levels of intelligence) on small devices. The Internet, static switched and mobile networks are converging around the TCP/IP model and Internet protocols are providing a framework for the deployment of new applications and services across this converged space. Internet Protocol (IP) enables the internetworking of disparate network nodes by providing a standardized addressing scheme and path determination techniques. Higher layer mechanisms can provide reliability, signaling and quality of service support. One potential outcome of these trends is the repartitioning of capabilities and responsibilities within a sensory network to a more distributed model as intelligence spreads outwards from the centre to the edge of the network. Sensory nodes can now be independent computing platforms capable of peer-to-peer communication and of interacting with interim network nodes in order to provide previously unavailable services. These could operate across a global inter-network and even between non-heterogeneous sensing nodes. Java provides a platform independent application development environment and network operating environment for code mobility and increasingly provides APIs and frameworks for internet protocols implementation and telecommunications and internetworking support. Although limitations still exist in the intelligent services supported across the Internet, new protocols and services are emerging which address these shortcomings. This paper seeks to discuss the potential impact of these developments on sensor networks.

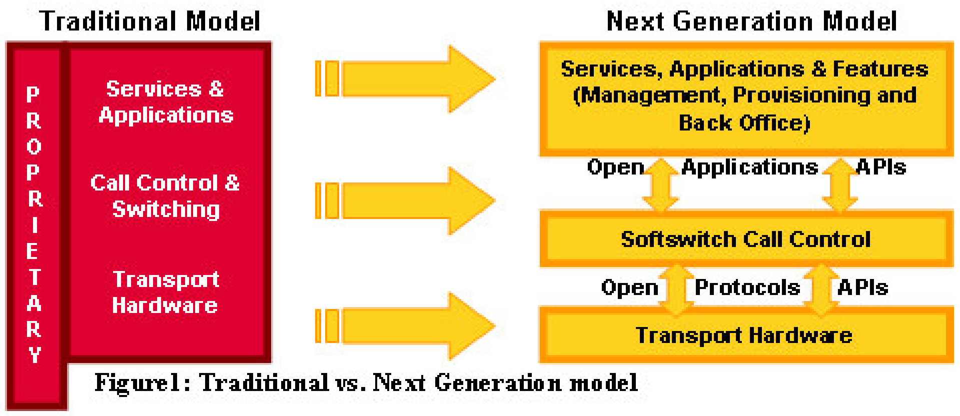


Figure1: Traditional vs. Next Generation model

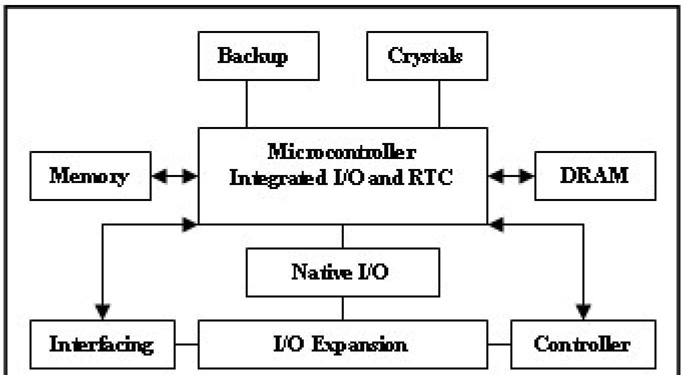


Figure 2: General architecture for an embedded system

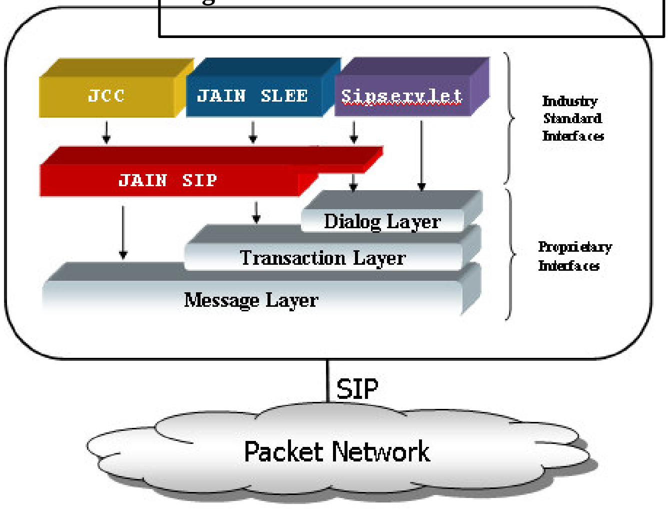
Modern Embedded Systems

An embedded system (*Figure 2*) is a device that contains programmed logic on a chipset that is used to control one or more functions of the device. It usually has more limited computational power, storage and interface functionality than a desktop platform. A real-time embedded system is often required to provide deterministic performance, often in a mission critical environment. This real-time behavior of embedded systems service logic is normally ***event-driven*** rather than conventional enterprise behavior. In comparison to enterprise computing, embedded systems use relatively thin components and perform lightweight asynchronous transactions at reasonably high frequencies. Unlike typical enterprise system business application logic, the most computationally intensive activities in embedded systems are generally more related to input/output (I/O) operations than database access related operations. The trend towards convergence in the networking world indicates that embedded systems will increasingly operate in a pervasive computing environment using wired and wireless transport technologies with IP connectivity for communication. General-purpose embedded systems with TCP/IP stack support have already emerged. Even with modest processing power, these can provide powerful event-driven distributed architectures with better functionality, scalability, availability, performance, manageability and security characteristics than ever before.

Suitability of Java (support for Internetworking)

Java is a relatively new language which has many advantages over other high-level programming languages, including: ***Simplicity***, ***Platform Independence***, ***Object Oriented, Multi-Threading, Robustness & Dynamic Binding*** *and* ***Security***. Many standard extensions and class libraries are provided in Java which supports the development of socket-based, client-server based and other distributed applications. Features like ports and servlets permit the rapid development of network applications based on Internet protocols while Remote Method Invocation (RMI) and the Java Messaging Service (JMS) support more complex distributed systems and can be used to effectively leverage distributed computational power to complete more complex tasks.

Figure3: JAIN Framework



The Java Intelligent Networks Framework (JAIN)

A key enabler in rapid application and service development and deployment is the availability of open and standard APIs that span NGN technologies but that abstract from the specifics of underlying data transport technologies. The JAIN framework is an extension of Java, and specifies a number of open and extendable Java technology APIs that support the rapid development of Next Generation communication-based products and services on the Java platform. Although JAIN is primarily a specifications framework, it has also provided a number of Application Package Interface (API's) that allow developers to access communications functions such as Call Control, Mobility Management and User Interaction in a signaling protocol-neutral way, while also providing more granular access to the underlying signaling protocols if needed. This can be achieved through high-level programming techniques. JAIN also defines a service creation environment and explicitly defines a Service Logic Execution Environment (SLEE). The SLEE concept is tightly mapped to event-driven services such as a Call Control, Alarming or other automated service and is eminently suitable for the execution of service logic and signaling in specialized event-driven engines. While SLEE is still at the specification stage, JAIN reference implementations for protocols such as SIP signaling moves Java into the carrier grade telecommunications domain today and promises much more for the future. From the perspective of embedded systems, the flexibility of the JAIN SIP API specification and the small size of the SIP toolkit identify this as a potentially rich application layer signaling solution for mobility management and other services. JAIN remains very much a work in progress, although we have been able to achieve a considerable amount using the SIP and other APIs. Nevertheless, aspects like performance, stability or failure situations have to be addressed and investigated properly before attempting to port the solution to embedded platforms.

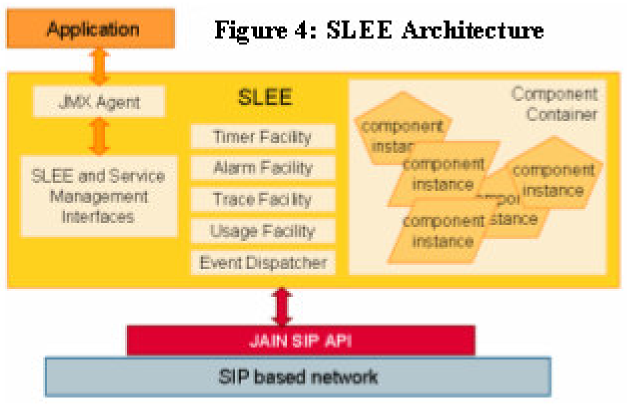


Figure 4: SLEE Architecture

JAIN SLEE

JAIN SLEE is high performance event processing platform suitable for event driven applications. It supports both simple and complex telecommunications applications. The SLEE framework is independent of underlying networks and it portable, robust and allows for reusable applications. (see *Figure 4*)

Rationale for the use of SIP

SIP is a signaling protocol and is not designed to be used to transfer data/media; it normally relies on protocols such as RTP/RTCP to transfer the media. This separation of signaling from the data exchange is an important characteristic that will make it possible to use different paradigms and modes of operation for signaling, control messaging and data transfer as appropriate. In typical SIP operation, SIP signaling is used to establish a transfer session, the session characteristics are negotiated between the end devices using SDP and the media is transferred using RTP/RTCP. There are a number of main features associated with SIP such as; ***Simplicity***, ***Scalability***, ***Flexibility/Extensibility***, ***Registration/Location***, Improved ***Security*** and ***Event Notification***.

Embedded System Support

The original Oak language from which Java derived was intended for embedded applications. The combination of platform independence and the adaptability of Java that allows it to work on micro-sized platforms by shedding non-essential code make it suitable for developing embedded system logic for a wide range of devices. Three main approaches currently exist for developing embedded systems applications: J2ME, Embedded Java and Personal Java. Two configurations are available: ***Connection Limited Device Configuration*** (CLCD), which is targeted at environments where 128-512Kb of memory is available for the Java environment and applications and ***Connected Device Configuration*** (CDC), which is targeted at environments, where more than 512Kb, usually about 2Mb, of memory is available for the Java environment and applications.

Java Real-Time Development

The Java Technology Model with Real-Time Extensions further leverages the capabilities of a real-time operating system (RTOS) to achieve the promise of hard real-time computing. Real-time systems are found in embedded applications as well as other applications that require a deterministic time behavior. The areas addressed in the Real-time extension (RTSJ) are; ***real-time threads***, ***asynchronous events***, ***interruptible non-blocking I/O***, access to ***physical memory***, ***scheduling***, ***garbage collection handling*** and ***timers***. The Java real time initiative is still very much a work in progress. Most embedded VM and hardware vendors seem to focus their efforts on J2ME and have no plan to implement RTSJ. RTSJ reference implementation is only a partial implementation and is not suitable for serious real-time applications.

Analysis

Our applied task is to create of remote sensor monitoring system that allows multiple independent sensor platforms to be monitored and be interrogated from any Internet station. The purpose of the system is to monitor the welfare of livestock in transport by monitoring the animal's heart-rate, temperature and environment factors. This system is composed of a number of embedded devices each connected to a specific senor or a mobile modem for connection to the outside world. The sensors are responsible for monitoring ambient and internal values. The remote monitoring system needs an embedded function to report back to a central site for permanent storage of data for scientific analysis. This reporting is going to be ***event-triggered***. An example of such an event would be if the sensor reading is outside a defined threshold or if the data file size has exceeded a predefined size. The collected data is sent back to the central site for permanent storage and analysis. Once the data is stored at the central site, the scientists are free to examine the data further and are able to correlate locational information with stress levels.

Implementation

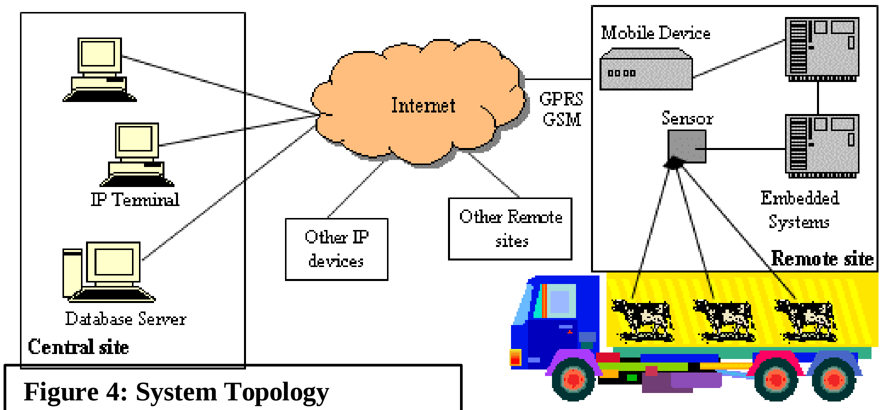


Figure 4: System Topology

The current system topology is composed of at least two sites. The remote site is made up of a number of IP nodes interconnected using Ethernet for LAN based connectivity and GPRS for WAN based connections. The remote site system is made up of a number of Java enabled embedded systems. The next section outlines a breakdown of the main core components.

1)Communication among embedded systems: The communication between the embedded systems is carried out using SIP J2ME. SIP is used to maintain data convergence among each of the connected devices, through the use of Instant Messages which are exchanged on a timed or triggered base. These Instant Messages are also routable outside the LAN through the use of public proxies. The goals of SIP J2ME are; it enables terminals supporting CLDC to run SIP enabled applications, it is firmly build and the CLDC Generic Connection framework. Another important factor is the API size small and to keep the number of created objects is at a minimum. This is very important in an embedded environment when memory and processor power is at a premium. **Instant Messaging Example:** Instant messaging is defined as the exchange of content between a set of participants in real time. Messaging between the nodes of a real-time mobile sensor network could be considered as an Instant Messaging application. Such an application could be implemented by using SIP but without requiring the establishment of a call. There is currently a proposal to extend the SIP specification by adding a new MESSAGE method. This method supports both the addressing and the transfer of any MIME type content between nodes but does not require prior call establishment. A MESSAGE request may traverse a set of SIP proxies using a variety of transport mechanism (UDP, TCP) before reaching its destination. The destination for each hop is located using the address resolution rules detailed in the SIP specifications. During traversal, each proxy may rewrite the request address based on available routing information. This method leverages Routing like functionality (the pre-pending of proxy information in this case) to provide a reply path. Provisional and final responses to the request will be returned to the sender as with any other SIP request. As a backup we have developed a Short Messaging Service (SMS) system. The difficulties of this system are that you are locked into a proprietary format and have little room for extensibility or interoperability and we feel that SIP provides far more reaching functionality, but as a backup SMS is a suitable medium.

Message F1	**Message F4**
`MESSAGE im:user2@domain.com SIP/2.0`	`SIP/2.0 200 OK`
`Via: SIP/2.0/UDP user1pc.domain.com`	`Via: SIP/2.0/UDP user1pc.domain.com`
`From: im:user1@domain.com`	`From: im:user1@domain.com`
`To: im:user2@domain.com`	`To:im:user2@domain.com;tag=ab8asdasd9`
`Call-ID: asd88asd77a@1.2.3.4`	`Call-ID: asd88asd77a@1.2.3.4`
`CSeq: 1 MESSAGE`	`CSeq: 1 MESSAGE`
`Content-Type: text/plain`	`Content-Length: 0`
`Content-Length: 30`	
`[1,105041071103,16.62,0,11,23]`	*Note that most of the header fields are simply reflected in the response. The proxy receives the response, strips off the top Via, and forwards to the address in the next Via, user1pc.domain.com and the result message is F4*

Table1: Instant Message exchange

2)Data retrieval from Sensors: The system is made up of a number of sensors which are primarily connected using open standard interfaces, which lead to the creation of a general package (API) for sensor reading and manipulation. The package is used for sensor polling, data storage and compression. Each of these attributes can be configured through the use of configuration files.

Table2: Configuration example from sensor package

`# serial port - that is connected to the sensor`	`# log file that will contains the readings`
`serial=serial0`	`log=duck_info.dat`
`# read for new data(specified in minutes)`	`# mechanism used for compression`
`read=1`	`compression=lzw,readings,ascii_table`
`# mechanism for exchange`	`# address of tftp server`
`exchange=tftp,ftp,tcp`	`tftp_server=10.1.1.100`
`# filename containing compressed data`	`# send data to server after x concurrent reads`
`tftp_file=test.zzz`	`send=7`
	`# sensor1 threshold`
	`threshold 37.0`

Once the data has been collected from the sensors it is converted into a generic PDU (protocol description unit). This generic PDU is very important because it means that all the data is in a common form and data processing is greatly reduced making it easier when the data is being permanently stored in the database at the central site. The PDU string contains not only the sensory data, but also a lot of meta-information about the sending device. The PDU is in the form of hexa-decimal octets or decimal semi-octets. The table below illustrates the PDU breakdown.

Table3: Sensor PDU breakdown

Octet(s)	**Description**
07 34 56	CRC (cyclic redundancy check)
99 30 92 51 61 95 80	Time stamp (semi-octets)
00	Sensor type: alarm/response/update
9B	Sensor ID/Sensory interval
FD	Data coding scheme/Data compression algorithm (00 – default coding/no compression)
0A	Length of payload.
E8329BFD4697D9EC37	Sensor data: 8-bit octets representing 7-bit data

The interval delay for polling for new data is set in the configuration descriptor as well as the defined threshold for that sensor and when the *readSensorValue()* is called the threshold levels are checked and if the threshold is breached then an *alarm type* Sensor Type is generated. It is reading is within the defined level then a *response type* is generated and the reading is logged. The final case is if the next reading has the same sensor reading then an update type is generated and the reading is logged.

Conclusions

We are entering a new phase in the development of distributed sensory networks. Embedded processing is becoming powerful enough to tackle an ever-widening range of applications. Wireless and wired networking is becoming ubiquitous, cheap, and low-power so that we can envision interconnecting all our embedded processors. Modern embedded systems can support either partial or entire TCP/IP stacks and will be inter-networked over the NGN using Internet protocols. This means that more distributed architectures will be possible in the area of mobile sensor networks, by leveraging emerging low-level access internet protocols such as SIP. The sheer breadth of scope of the Java initiative seems set to encompass the previously disparate areas of open-standard internetworking, embedded systems, real-time logic and high-level service development making it a key ingredient in a converging technologies world. ***An embedded device is becoming just another IP node on a mobile network***. IP Communication and open protocols are the crux of NGN's, investment and converged applications. Java provides convergence at the application level and it increasing possible for non-specialist 3rd parties to create value added sensor system services without detailed knowledge of the underlying network complexities.

Looking forward

We are currently working on data retrieval using a SIP based call from the sensor network to the central site and to further leverage the capabilities of Instant Messaging enabling them to be routed through public proxies to be received on any IP enabled device running the SIP stack. We are also investigating the feasibility of the transfer of Real-time video across the GPRS backbone encapsulated in a RTP (real-time protocol) socket from the sensor network to the central site.

References

1. Arjun Roychowdhury & Stan Moyer, Instant Messaging and Presence for SIP Enabled Networked Appliances, 2002
2. Wolfgang Kellerer, Intelligence on Top of the Network: SIP based Service Control Layer Signaling, 2002
3. Phelim O'Doherty, Java Technology for Internet Communications, 2003
4. M.Satyanarayanan, Pervasive Computing: Vision and Challenges, 2001