CONNECTING SENSORS AND ROBOTS THROUGH THE INTERNET BY INTEGRATING MICROSOFT ROBOTICS STUDIO AND EMBEDDED WEB SERVICES

Oscar Almeida
Texas A&M University
College Station, TX 77843
oscar10@tamu.edu

Johannes Helander
Microsoft Research
Redmond, WA 98052
jvh@microsoft.com

Henrik Nielsen
Microsoft
Redmond, WA 98052
henrikn@microsoft.com

Nishith Khantal
Indian Institute of Technology, Kanpur
nkhantal@gmail.com

ABSTRACT

While sensors and robots are prevalent in modern electronic devices, there still exists a general lack of interoperability between them. It would be nice if all real world objects could speak the same language, regardless of their differing properties. This work combines the development of two software platforms, Microsoft Invisible Computing and Microsoft Robotics Studio, to provide the means for a real-time sensor network to achieve this goal. Microsoft Invisible Computing, a light-weight operating system and communications middleware compatible with several existing hardware platforms, allows real-time communication with sensors and robots, along with the means to send and receive XML/SOAP messages. Microsoft Robotics Studio, on the other hand, can provide the decentralized orchestration, simulation and user interface necessary for sensor network nodes to interoperate efficiently. Ultimately, this work lays a foundation for a world of compatible objects.

KEYWORDS

XML, Services, Robots, Embedded Systems

1. INTRODUCTION

Today, new technologies are continuously emerging with embedded sensors and robots. This includes, but is not limited to, consumer devices and medical tools in hospitals. A property uniting most of these devices is that they are incompatible with each other. That is to say, that each group of devices has a somewhat unique method of communication with other devices.

To clarify the scope of this work, we are targeting small embedded devices that can perform their functions without even being noticed by the end user. As little as one to two-dollar processors can drive these devices. With memory, wireless capabilities, and a sensor also added, the total estimated cost for such devices can be as low as $25.

The primary scenario used in this paper is patient monitoring in a hospital. Here there are hundreds of tools, sensors, and robots, few of which share the same communication protocol. Imagine representing these objects by web services, each having a common notion of name, some states, and a list of other devices that they need to talk to. Then on an even broader level than the network nodes themselves could be orchestration and fusion of such objects to provide evidence to make assumptions about the safety of a patient. For example, a patient may drop a walking stick with fall-detecting capabilities [13]. Normally this would trigger an alarm of some sort. But what if the patient's bed contained a sensor that could tell if the patient was on the bed? If the stick fell, but the patient had just gotten into bed, then there was in fact no real emergency. A lower priority alarm can be sent to help the patient restore the fallen stick to be ready for use.

Embedded Web Services (EWS), a component of Microsoft Invisible Computing, enables such devices to become sensor network nodes. Included is the ability to communicate with a sensor or robot, along with the capability to communicate with other nodes. Microsoft Robotics Studio, a PC-based development tool for easy creation of robotics applications, can be used to manage the sensor nodes in a decentralized, user-friendly manner. The remaining sections include introductions to the Embedded Web Services and Robotics Studio, discussion on related work, followed by our design, implementation, and results.

2. EMBEDDED WEB SERVICES AND ROBOTICS STUDIO

Embedded Web Services realizes the ability of an embedded system to become a sensor network node. By transmitting and receiving SOAP messages, multiple nodes are able to intercommunicate, even using different web service protocols. A major advantage of this platform in addition to network functionality is the ability to monitor sensors in real-time. Thus, a sensor can be polled, for example, and talk to other network nodes as well [12].

Robotics Studio is a development tool allowing easy creation of robotics and sensor-based applications. It provides a decentralized, asynchronous environment for managing web services. The communication between services occurs via Decentralized Software Services Protocol (DSSP) [11], a superset of the HTTP methods. This protocol allows for structured data retrieval, monitoring, and manipulation. Robotics Studio also contains a 3-d physics engine and simulation environment. Here real and simulated sensors can be mixed and matched as desired. In fact, Robotics Studio does not actually know the difference between the two.

Although Microsoft Invisible Computing provides the means to communicate with sensors in real-time and transfer data to other network nodes, the scope of each node is mostly reduced to itself. That is, overall system management is not provided for complex distributed tasks. Furthermore, it is common for network nodes to require as little computation as possible to meet timing and size requirements. While EWS has some orchestration support, it is not practical to do sophisticated modeling or planning on microcontroller nodes with limited computation power. Robotics Studio, running on PC class computers, can provide decentralized management of network nodes with very little overhead. Thus, leaving the individual nodes to EWS and the overall system management to MSRS makes an efficient and effective combination.

Robotics Studio is also capable of having a user interface that cannot be achieved in most embedded systems. The 3-D graphics and physics engines provide a method of displaying the status of the network nodes actuated by EWS. Take for instance the hospital scenario again. A 3-D floor plan model of a hospital can be created in Robotics Studio. Visualizing the locations and states of several sensors in the hospital can be very beneficial for patient monitoring and for responding to alarms.

One advantage of the combination of Robotics Studio and EWS in this medical scenario is the removal of false positives. Microsoft Invisible device could monitor a sensor such that it over-detects potential accidents. While over-detecting alarms may remove the possibility of false negatives, false positives may likely become present. Robotics Studio could then use evidence from other sensors to detect and remove false positives.

The ability to mix simulated and real sensors in Robotics Studio can also prove to be useful for system testing. If certain devices are not ready to be directly monitored, simulation for such devices can be used for testing the devices that are already monitored by Robotics Studio. The 3-D physics engine allows identical output between a simulated and actual robot, since Robotics Studio cannot tell the difference.

Alternatively, an entire simulation environment could be created to provide a base for real sensors to be verified against.

3. RELATED WORK

Significant work, on platforms similar to those targeted by Microsoft Invisible Computing, has also been done recently in the field of sensor networks. [1] presents a sensor network testbed that uses a centralized server for reprogramming nodes, data logging, and supplying a web interface. This allows for a reduced amount of attention required by the individual nodes, since all can be accessed by a conveniently central location. [7] on the other hand explains an asynchronous approach to designing sensor networks for distributed applications. Comparisons to scheduled networks are also made. [2] describes a sensor network monitoring software that, like EWS, allows different types of sensors to be integrated and communicate via XML through the internet. Because it is written in Java, this software achieves OS platform independence, while requiring somewhat more powerful hardware. While our system also uses XML, it is in the form of SOAP messages. With this method, we can be careless of what types of objects are being added to the system. In other words, there does not need to be a global library of acceptable object types. Only at the node level should it be known what types of objects can be interacted with. [3] involves a real-time embedded OS designed with scheduling and timing advantages over a number of other existing embedded operating systems. [4] constructs an interpretation model for sensor data and queries. The work described in [5] discusses several issues regarding important real-time issues to be considered when designing complex robotics systems and applications. [6] emphasizes the reality of heterogeneous sensor networks, in which each node may operate on different platforms. A combination of simulation and emulated sensors being controlled one instruction at a time is also discussed. [8] also involves heterogeneous sensor networks but instead focuses on the appropriateness of TCP/IP as a means of communication between network nodes. The work performed in [9] suggests that mobile sensor networks can be more appropriate at times, depending on what is being monitored. Lastly, [10] presents a sensor network architecture with the purpose of intertwining network communication and computing operations. Various protocols and middleware options are discussed as well.

4. DESIGN

The web service architecture that we integrated into the Microsoft Invisible platform can be divided into five layers: connector, converter, DSSP interface, resource, and property class. The connector manages network connections and handles TCP/IP and HTTP processing. The converter takes XML and converts it to native representation based on the schema of the data. The DSSP interface implements the parameters necessary in the incoming and outgoing SOAP messages. This implementation supports customized data types by using three special parameters provided by the converter. The data value is referenced by a pointer, while a string specifies the type, and an integer stores the data size. Several DSSP functions share code by calling the same function in the resource layer.

We designed the resource layer to consist of a single directory tree of resource objects. Each object has a value and a property class. Part of the state defined by the property class, for example a "name" field, is used to identify the resources in the directory tree. Every resource represents either a service or a service's state. Resources that act only as a subdirectory are represented by services. The identification is used for searching purposes, while the value and property class define the object. Again, the data is merely pointed to, whereas the resource type and method interface are stored in the property class.

All property class methods (e.g. Copy, Delete, and Apply) are shared, but not necessarily implemented, by all resources. It is in this layer that resources are found, created, destroyed, stored, and retrieved. Additionally, a resource may apply another resource (e.g. policy) to itself to validate its actions. For example, a resource storing sensor data may only report its data to others if the data is above some lower threshold applied to it.

Code reuse is heavily emphasized to save space. The DSSP functions CREATE, DELETE, DROP, INSERT, REPLACE, UPDATE, and UPSERT all utilize the same resource-changing method in the resource layer. Likewise, the functions GET, LOOKUP, and QUERY all use a resource-enumerating method that returns a list of resources at a certain level in the resource directory. The enumeration involves creating a list of pointers to the desired resources. Then each resource's value is pulled from the list, one by one, being added to the output list if it passes the filter. GET would return all resources at a given level, while LOOKUP would return only services, and QUERY would return some specific resource.

Subscriptions and policies have slightly different implementations from the above. When a subscription to resource X is received, a directory path identical to that of X is created under the "root/Subscriptions" directory. Here the subscriber, subscription expiration, and notification count are stored. Anytime resource X is changed thereafter, its corresponding subscriber will be sent a notification. Policies, such as thresholds and trust scripts, are handled in an identical fashion but in the "root/Evidences" directory.

5. IMPLEMENTATION

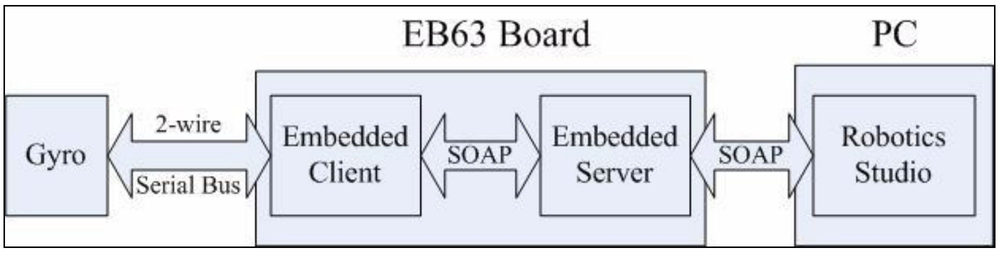
The following demo was performed in an outpatient monitoring medical context. The angular velocity of a walking stick prototype, intended for use by the elderly, is measured by an embedded gyro sensor. Such measurements can be used to detect when the user may have fallen and generate an alarm to the medical personnel. The scope of this demo is to poll the gyro with a microcontroller and send the data over the network to Robotics Studio using DSSP.

The microcontroller used in the demo is the Atmel EB63 Arm7 board, which runs at 25MHz. This board is loaded with the Microsoft Invisible Computing platform. Tasks include polling the gyro and managing the web services used to communicate with Robotics Studio. Because the actual board has no Ethernet port, the Ethernet packets generated by the network protocol are sent via a serial line instead to the PC and are then forwarded using a Virtual NIC driver. All in all the PC here simply serves as a very bulky Ethernet card.

The gyro used is the Gyration MG1101 MicroGyro. It contains two axes of measurement and operates over a 2-wire serial bus. In this project, we placed the gyro inside and at the base of the walking stick prototype, shown at right. The gyro measures the angular velocity of the walking stick.

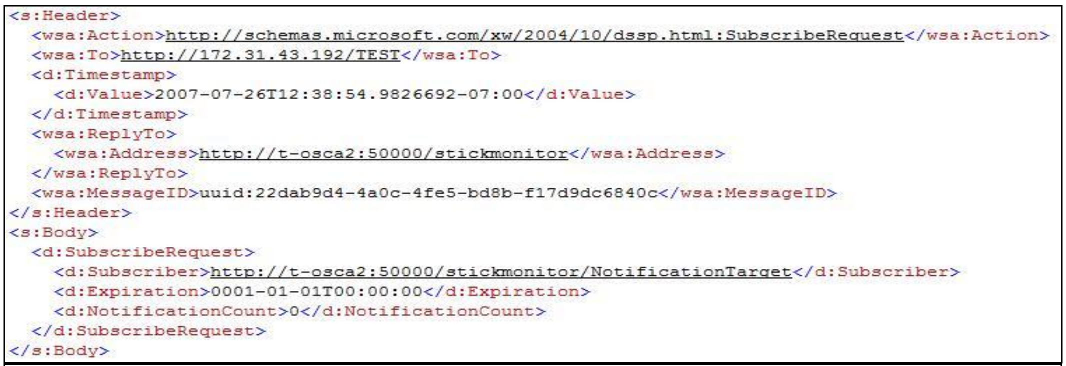
Three parties are involved in the communications, as shown in figure 1. "Embedded Client" is a local application running on the board, which is the software that actually controls the gyro. It communicates locally with the "Embedded Server" via the loopback network interface using SOAP. "Robotics Studio" runs on a PC and communicates with "Embedded Server" using the Ethernet link.

Figure 1. The communication between the gyro sensor, Embedded Client, Embedded Server, and Robotics Studio is shown here. The gyro is connected via 2-wire serial bus to the microcontroller board. The Embedded Client, Embedded Server, and Robotics Studio talk via SOAP messages

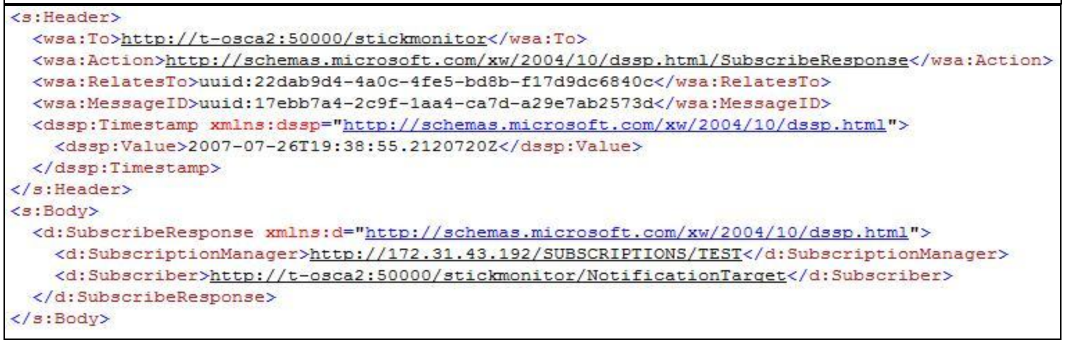


At the start, Embedded Client asks Embedded Server to create a web service for itself. This service is called 'TEST' and will serve to hold the gyro data. Embedded Client then inserts the initial state of the gyro into the 'TEST' web service on Embedded Server. At this point, Robotics Studio subscribes to the service 'TEST', to be notified of any changes in the state of the gyro. The message exchange is shown in figure 2.

Figure 2. The top xml snippet below shows the subscribe request by Robotics Studio to the Embedded Server. The significant data is in the body of the message. The Embedded Server stores this information and then sends a response, shown in the lower xml code block



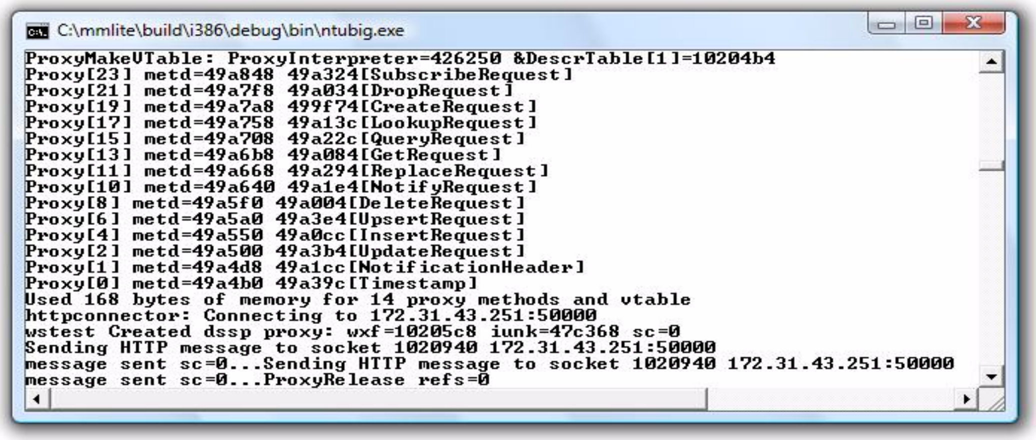
```xml
<s:Header>
  <wsa:Action>http://schemas.microsoft.com/xw/2004/10/dssp.html:SubscribeRequest</wsa:Action>
  <wsa:To>http://172.31.43.192/TEST</wsa:To>
  <d:Timestamp>
    <d:Value>2007-07-26T12:38:54.9826692-07:00</d:Value>
  </d:Timestamp>
  <wsa:ReplyTo>
    <wsa:Address>http://t-osca2:50000/stickmonitor</wsa:Address>
  </wsa:ReplyTo>
  <wsa:MessageID>uuid:22dab9d4-4a0c-4fe5-bd8b-f17d9dc6840c</wsa:MessageID>
</s:Header>
<s:Body>
  <d:SubscribeRequest>
    <d:Subscriber>http://t-osca2:50000/stickmonitor/NotificationTarget</d:Subscriber>
    <d:Expiration>0001-01-01T00:00:00</d:Expiration>
    <d:NotificationCount>0</d:NotificationCount>
  </d:SubscribeRequest>
</s:Body>
```



```xml
<s:Header>
  <wsa:To>http://t-osca2:50000/stickmonitor</wsa:To>
  <wsa:Action>http://schemas.microsoft.com/xw/2004/10/dssp.html/SubscribeResponse</wsa:Action>
  <wsa:RelatesTo>uuid:22dab9d4-4a0c-4fe5-bd8b-f17d9dc6840c</wsa:RelatesTo>
  <wsa:MessageID>uuid:17ebb7a4-2c9f-1aa4-ca7d-a29e7ab2573d</wsa:MessageID>
  <dssp:Timestamp xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html">
    <dssp:Value>2007-07-26T19:38:55.2120720Z</dssp:Value>
  </dssp:Timestamp>
</s:Header>
<s:Body>
  <d:SubscribeResponse xmlns:d="http://schemas.microsoft.com/xw/2004/10/dssp.html">
    <d:SubscriptionManager>http://172.31.43.192/SUBSCRIPTIONS/TEST</d:SubscriptionManager>
    <d:Subscriber>http://t-osca2:50000/stickmonitor/NotificationTarget</d:Subscriber>
  </d:SubscribeResponse>
</s:Body>
```

Now the system is initialized and enters the operational state. The EB63 board polls the gyro at a set rate, and accordingly Embedded Client updates the state of the gyro in Embedded Server. This update triggers the creation of a proxy between Embedded Server and Robotics Studio. Once the connection is established, the updated gyro measurements are sent to Robotics Studio. The Robotics Studio web service collecting this data (gyro measurements) can be viewed in a web browser in real time. Below is the printed debug information displayed when Embedded Server creates the proxy to Robotics Studio and sends it a message.

Figure 3. The debug information display that occurs at the time of proxy creation between the Embedded Server and Robotics Studio

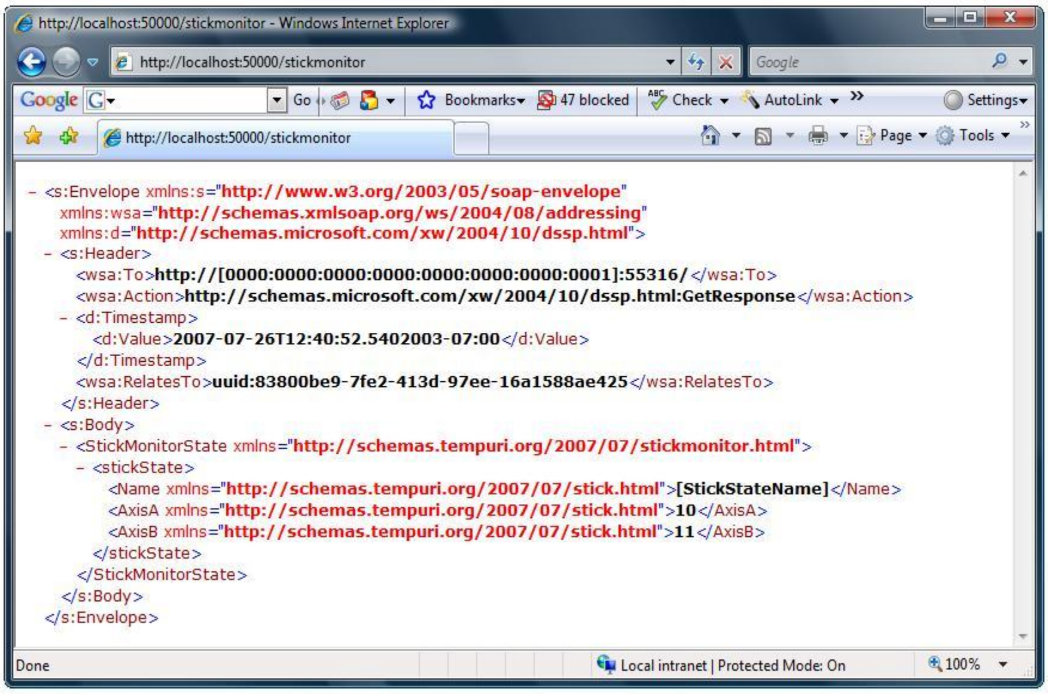


```
ProxyMakeVTable: ProxyInterpreter=426250 &DescrTable[1]=10204b4
Proxy[23] metd=49a848 49a324[SubscribeRequest]
Proxy[21] metd=49a7f8 49a034[DropRequest]
Proxy[19] metd=49a7a8 499f74[CreateRequest]
Proxy[17] metd=49a758 49a13c[LookupRequest]
Proxy[15] metd=49a708 49a22c[QueryRequest]
Proxy[13] metd=49a6b8 49a084[GetRequest]
Proxy[11] metd=49a668 49a294[ReplaceRequest]
Proxy[10] metd=49a640 49a1e4[NotifyRequest]
Proxy[8] metd=49a5f0 49a004[DeleteRequest]
Proxy[6] metd=49a5a0 49a3e4[UpsertRequest]
Proxy[4] metd=49a550 49a0cc[InsertRequest]
Proxy[2] metd=49a500 49a3b4[UpdateRequest]
Proxy[1] metd=49a4d8 49a1cc[NotificationHeader]
Proxy[0] metd=49a4b0 49a39c[Timestamp]
Used 168 bytes of memory for 14 proxy methods and vtable
httpconnector: Connecting to 172.31.43.251:50000
wstest Created dssp proxy: wxf=10205c8 iunk=47c368 sc=0
Sending HTTP message to socket 1020940 172.31.43.251:50000
message sent sc=0...Sending HTTP message to socket 1020940 172.31.43.251:50000
message sent sc=0...ProxyRelease refs=0
```

The notifications are one-way SOAP messages, meaning that they will not wait for a response from the target. A notification can be configured in Robotics Studio in the form of any of the state changing commands (REPLACE, INSERT, UPDATE, and UPSERT). For this demo, the notification is the equivalent of a Replace call, meaning that the entire state kept in Robotics Studio will be replaced by the entire body of the incoming message.

Shown below is the simple user interface of this demo. While Robotics Studio is running, a user may simply open a web browser to the URL of the Robotics Studio service. If the service state is desired, the web browser will perform a Get function call on the service and display the response. Here the measurements on each axis of the gyro are shown in the body of the message. Of course, a much more detailed interface could be created in Robotics Studio using the 3-D physics engine.

Figure 4. The simplest Robotics Studio interface, which is simply a page in a web browser at the service's URL. The web browser performs a GET operation on the service of interest and displays the data it contains


http://localhost:50000/stickmonitor - Windows Internet Explorer
http://localhost:50000/stickmonitor
Google
Go
Bookmarks
47 blocked
Check
AutoLink
Settings
Page
Tools
Done
Local intranet | Protected Mode: On
100%

```
- <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:d="http://schemas.microsoft.com/xw/2004/10/dssp.html">
  - <s:Header>
      <wsa:To>http://[0000:0000:0000:0000:0000:0000:0000:0001]:55316/</wsa:To>
      <wsa:Action>http://schemas.microsoft.com/xw/2004/10/dssp.html:GetResponse</wsa:Action>
    - <d:Timestamp>
        <d:Value>2007-07-26T12:40:52.5402003-07:00</d:Value>
      </d:Timestamp>
      <wsa:RelatesTo>uuid:83800be9-7fe2-413d-97ee-16a1588ae425</wsa:RelatesTo>
    </s:Header>
  - <s:Body>
    - <StickMonitorState xmlns="http://schemas.tempuri.org/2007/07/stickmonitor.html">
      - <stickState>
          <Name xmlns="http://schemas.tempuri.org/2007/07/stick.html">[StickStateName]</Name>
          <AxisA xmlns="http://schemas.tempuri.org/2007/07/stick.html">10</AxisA>
          <AxisB xmlns="http://schemas.tempuri.org/2007/07/stick.html">11</AxisB>
        </stickState>
      </StickMonitorState>
    </s:Body>
  </s:Envelope>
```

6. RESULTS

Although the most significant results of this work involve interoperability between sensors and robots through the Internet, three quantitative measurements were performed. The first set of measurements consisted of the time taken to create web services on the Embedded Server. Below are the time requirements for creating the service directory in Table 1. Notice that the time to create services increases as the depth in the tree increases. This is because an inserted service's parent must first be located.

Table 1. The time needed to create the services below. A lower threshold policy is inserted into EVIDENCE/TEST0/TEST/TEST2 while TEST0/TEST/TEST2 contains actual gyro measurements

Directory Location	Time taken to create service (ms)
EVIDENCE	66.53 ms
EVIDENCE/TEST0	80.21 ms
EVIDENCE/TEST0/TEST	89.27 ms
EVIDENCE/TEST0/TEST/TEST2	96.68 ms
TEST0	84.03 ms
TEST0/TEST	86.25 ms
TEST0/TEST/TEST2	111.97 ms

The next set of measurements was simply a summation of the sizes of the files necessary to run the demonstration program. The table below displays the individual sizes of several file groupings. It is worth noting that the total ROM size is less than 256KB. The files in Table 2 are ordered from the lowest to highest abstraction levels. "BASE" to "DRIVERS" are very hardware specific, while "NET" to "HTTP" are network intensive and constitute the connector. "XML" to "SOAPMETA" contain the converter and related middleware files, and remaining files are directly accessible by the user applications. DSSP implements the DSSP protocol. DSSP and an optional WS-Management component (not included) take use of the CIMDB files, which contains the "resource layer" from section 4 of this paper. The DSSP files are the major addition of this work to the Microsoft Invisible platform. Several other files experienced minor changes to support this new protocol.

Table 2. The categorical listing of the file sizes, in bytes, included in a working implementation of this demonstration. This list is expandable to include more functionality from Microsoft Invisible Computing

File	ROM	RAM	# Files
BASE	22928	292	50
MACHDEP	5944	1632	22
CRT	14372	32	59
DRIVERS	11348	308	23
NET	57728	1844	67
TCP	13276	76	10
DHCP	5576	96	4
HTTP	21292	168	15
XML	12708	16	8
CONVERTER	37644	612	19
SOAP	17792	348	39
SOAPMETA	16112	20	26
DSSP	10636	144	5
CIMDB	12040	80	23
TOTAL	**259396**	**5668**	**370**

Lastly are the memory allocation measurements and thread count at different points in the demo application process. The column labeled "Command" shows what was entered in the command line interface for initializing the Microsoft Invisible Platform. Such instructions were performed to set up the network interface, establish a DHCP IP address for the "Embedded Server", and to actually run the demo application.

Table 3. The number of bytes in use at each stage of the demonstration application is shown below, as are the number of active threads at each step

Command	Memory Used (Bytes)	# Active Threads
[Initial]	11,056	1
protocol.cob start	46,464	6
start protocol.cob dhcp	53,376	8
http.cob	61,176	9
protocol.cob lo0 127.0.0.1	61,040	7
cimdb.cob	55,144	7

7. FUTURE WORK

Survey of more hardware, appropriately smaller sensors, microcontrollers, wireless, GPS, etc. is necessary to make a more concrete example of this work. Additionally, further development of a Microsoft Robotics Studio user interface application in conjunction with Embedded Web Services is also a major goal. This includes defining how ubiquitous services can be asynchronously well-managed, monitored, and discovered.

The capabilities of Robotics Studio graphically are far beyond the scope of the demonstration presented in this paper. The scenario of hospital patient monitoring by combining EWS and Robotics Studio is a specific future research project being considered. We plan to create a visual hospital setting where multiple sensors are fused by Robotics Studio to provide information that the nodes cannot provide on their own.

8. CONCLUSION

The significance of this work is the idea that all real sensors and robots can be represented by web services. Once such an abstraction is made, Embedded Web Services can be used to represent individual nodes while Microsoft Robotics Studio orchestrates them. Not limited to medical contexts, this work provides the means of interoperability that current technology often lacks. Ultimately, the dream of global compatibility and communication is realized.

As a result of the work presented in this paper it is now possible to use DSSP and services as a generic substrate for sensor networks, or the next generation robotic networks. The integration of the physical modeling and orchestration capabilities of the Microsoft Robotic Studio makes it possible to quickly build simulated environments that can be partially or completely physically realized. The Embedded Web Services make it practical to build microcontroller based devices with limited capabilities without having to compromise on interoperability or having to resort to proxy computers. A common data serialization format obviates the need for most dedicated protocols and makes it easy to turn physical objects into services.

REFERENCES

[1] Werner-Allen, G., et. all, 2005. MoteLab: A Wireless Sensor Network Testbed. *Fourth International Symposium on Information Processing in Sensor Networks.* Los Angeles, California, USA, pp. 483-488.

[2] Yu, M., et. all, 2007. NanoMon: A Flexible Sensor Network Monitoring Software. *The 9th International Conference on Advanced Communication Technology.* Gangwon-Do, pp. 1423-1426.

[3] Park, S., et. all, 2006. Embedded Sensor Networked Operating System. *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing.* Gyeongju, South Korea, 5 pp.

[4] Imai, M., et. all, 2006. Semantic Sensor Network for Physically Grounded Applications. *9th International Conference on Control, Automation, Robotics, and Vision.* Singapore, pp. 1-6.

[5] Buttazzo, G., 1996. Real-time Issues in Advanced Robotics Applications. *Proceedings of the Eighth Euromicro Workshop on Real-Time Systems.* L'Aquila, Italy, pp. 133-138.

[6] Polley, J. et. all, 2004. ATEMU: A Fine-grained Sensor Network Simulator. *First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks.* Santa Clara, California, USA, pp. 145-152.

[7] Narayanan, S. And Jones, D., 2005. On the Performance of Asynchronous Sensor Networks for Detection. *Proceedings of the 2005 International Conference on Intelligent Sensors, Sensor Networks and Information Processing.* Melbourne, Australia, pp. 241-246.

[8] Lei, S. et. all, 2006. Connecting Heterogeneous Sensor Networks with IP Based Wire/Wireless Networks. *The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquuitous Systems.* Gyeongju, Korea, 6 pp.

[9] Chin, T., et. all, 2005. Exposure for Collaborative Detection Using Mobile Sensor Networks. *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference.* Washington D.C., USA, 8 pp.

[10] Abdelzaher, T., et. all, 2007. Towards a Layered Architecture for Object-Based Execution in Wide-Area Deeply Embedded Computing. *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*. Santorini Island, pp. 133-140.

[11] Nielsen, H. and Chrysanthakopoulos, G., 2007. Decentralize Software Services Protocol – DSSP/1.0. [Online] http://download.microsoft.com/download/5/6/B/56B49917-65E8-494A-BB8C-3D49850DAAC1/DSSP.pdf.

[12] Helander, J., 2005. Deeply Embedded XML Communication: Towards an Interoperable and Seamless World. *Proceedings of the 5th ACM International Conference on Embedded Software.* Jersey City, NJ, USA.

[13] Almeida, O. et. all, 2007. Dynamic Fall Detection and Pace Measurements in Walking Sticks. *Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Plug-and-Play Interoperability.* Cambridge, MA, USA, 3 pp.