Extending the Grid to Support Remote Medical Monitoring

Carl Barratt[5], Andrea Brogni[8], Matthew Chalmers[2], William R. Cobern[6], John Crowe[5], Don Cruickshank[7], Nigel Davies[3], Dave De Roure[7], Adrian Friday[3], Alastair Hampshire[4], Oliver J. Gibson[6], Chris Greenhalgh[4], Barrie Hayes-Gill[5], Jan Humble[4], Henk Muller[1], Ben Palethorpe[5], Tom Rodden[4], Chris Setchell[1], Mark Sumner[5], Oliver Storz[3] and Lionel Tarassenko[6]

[1]*Department of Computer Science, University of Bristol*

[2]*Computing Science, University of Glasgow*

[3]*Computing Department, Lancaster University*

[4]*School of Computer Science & IT, University of Nottingham*

[5]*School of Electrical and Electronic Engineering, University of Nottingham*

[6]*Department of Engineering Science, University of Oxford*

[7]*Department of Electronics and Computer Science, University of Southampton*

[8]*Department of Computer Science, University College London*

Abstract. In this paper we show how we have used and adapted GT3 to support scalable and flexible remote medical monitoring applications on the Grid. We use two lightweight monitoring devices (a java phone and a wearable computer), which monitor blood glucose levels and ECG/SpO2 activity. We have connected those devices to the Grid by means of proxies, allowing those devices to be intermittently connected. The data from the devices is collected in a database on the Grid, and practitioners can obtain real time data or observe the patients historical data.

Introduction

The emergence of e-science and initiatives such as the UK e-science programme has been driven from the initial suggestion of "the Grid" as a distributed computing infrastructure for advanced science [1]. We have already seen considerable progress on the construction of such an infrastructure with software facilities such as the Globus Toolkit [2] becoming freely available based on the premise that high bandwidth communication allows storage and computational resources to be shared by a range of scientists accessing these services from their labs. Initiatives surrounding the GGF [3] such as OGSA [4] and work on the semantic grid [5] and OGSA-DAI [6] outline a clear position where powerful computational services and very large amounts of data are readily available to a distributed community of scholars.

However, not all science takes place within the research lab and when we consider those areas where a significant portion of scientific activity takes place away from the lab a mismatch is evident between the provision of services enabled by current grid technology and the needs of the scientist in the field. Currently, the link between the field scientist undertaking remote work "on site" and their home lab is poorly supported and is often a significant bottleneck in the scientific process.

Within the MIAS/ Equator Medical Devices Project we wish to address this potential scientific bottleneck by considering the development of mobile access to Grid services and how these services may be connected to a heterogeneous collection of mobile devices. Mobile access to grid facilities requires significant research to tack to two core research challenges.

- ***Making remote data available to the Grid*** in order that a wider scientific community can access scientific data as quickly as possible often using varying quality communication services.
- ***Making Grid facilities available to remote users*** when these need to be delivered across lower bandwidth communication using devices with significant display and processor limitations.

Our particular interest is in solving the core technological problems involved in extending the grid by exploring these challenges within the medical domain. The need to make Grid facilities available "in the field" is particularly critical in the case of medical services where much of the day to day work of medicine centres on the patient requiring a number of medical professionals to correlate medical data with patient examination and observation.

Grid technology and medical devices

The current trend towards Telemedicine and Telecare evident in the UK [7] has seen an explosion in the range of locations where advanced medical care needs to be delivered. E-medicine initiatives such as NHSdirect have illustrated the need to maximise the flexibility of delivery of health care. The ultimate goal is to increase the availability of medical care in order to both reduce the demands on hospital services and to improve the long term care and recovery of patients.

Existing trials in Telemedicine and Telecare such as those carried out by the Oxford centre for e-health [7], the Biomedical Informatics group at Nottingham University [8] and the Glasgow Royal

Infirmary and Glasgow University [9] have demonstrated the feasibility of remotely monitoring patients as part of an overall care programme or as part of a clinical trial. However, these efforts have tended to be small scale in nature and have typically required the development of bespoke sensors and purpose build infrastructure for the logging of data for analysis.

In this paper we present our initial work in creating a medical monitoring infrastructure that exploits standard grid software suitable for future clinical trials. The developed infrastructure allows lightweight medical devices to be made available on the grid as Grid Services. We have developed two distinct medical devices based on this infrastructure: a wearable medical monitoring jacket which independently delivers medical data to the Grid for analysis by researchers and clinicians; and a Java phone based blood glucose device which allows patients to self report medical information onto the Grid.

Device 1: The Monitoring Jacket

In developing our first medical device we have used a standard wearable platform (the Cyberjacket [10] developed at Bristol), which comes with wireless connectivity, and augmented it with three purpose built sensors: an ECG, an Oxygen Saturation Monitor and a temperature sensor. This augments the standard sensors on the Cyberjacket for position sensing (using GPS), and motion sensing (using accelerometers).

Whereas current state of the art monitoring systems which typically require purpose built devices or significant customisation, the Cyberjacket has a modularised architecture. This means that health researchers can easily customise a jacket for their experiment. It also decreases the cost of units, as they are reusable in different configurations.

Wearable architecture

The wearable system consists of an ADS `bitsy' processor unit (based on the StrongARM), with a custom 9-wire bus embedded in the fabric of the jacket. The bus provides sensors with power, ground, and communications via three serial links. Two links run at RS-232 levels, and are for dedicated RS-232 devices. Any commercially available device that runs RS-232 (such as a GPS) can be connected to one of these busses. The third bus runs at TTL level, at 4800 baud, and is used as a drop-link bus on which tens of devices (e.g. medical sensors) can be connected. In addition, the bitsy offers stereo sound I/O that can be used to give feedback to the wearer.

The standard sensors can be attached to measure a range of activities: a compass mounted on a pair of headphones measures the direction in which the wearer is looking, and an accelerometer mounted vertically on the back can detect walking. A typical configuration of the medical wearable is shown in figure 1.

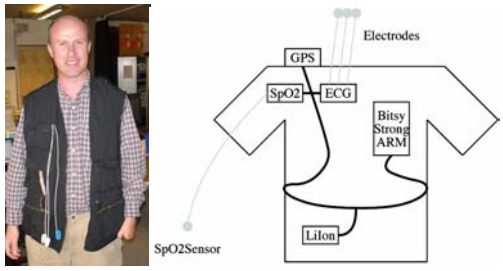

Figure 1: A typical configuration of the wearable

Sensor architecture

Typically, information is gathered from multiple sensor types in order to more accurately track and diagnose a given disease state or improve the data fed into clinical trials. The combination of electro-physiological and other medical parameters that must be monitored will vary for a specific medical condition. Consequently we have designed modular sensors units that can be easily added or removed to facilitate customisation for a particular trial. At the time of writing, the sensor modules include:

- **Blood oxygen saturation (SpO2)** derives the level of oxygen present in arterial blood by calculating the ratio of light absorption resulting from oxygenation and reduced haemoglobin through a well-perfused body part (the finger).
- **3 Point Electrocardiogram (ECG)** monitors the signal produced on the surface of the skin by the electrical activity of the heart.
- **Skin and ambient temperature** are monitored using clinical grade thermistors.

Each sensor uses analogue techniques to obtain, non-invasively, signals that impart rich medical information about the patient. The signals are conditioned, sampled and processed prior to assembly into packets of digitised data. All sensors conform to a similar packet format, which includes a header (depicting the origin of the packet and the type of data included within it), the data itself and some indication of the validity of the data.

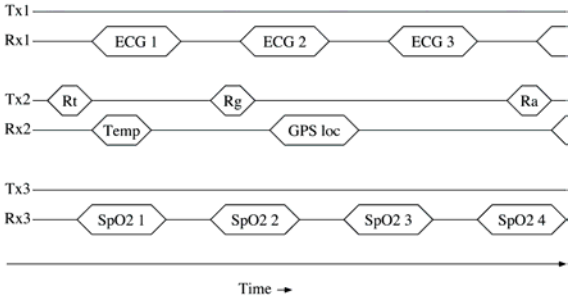

Figure 2: The wearable bus protocol

The packets of data generated by a particular sensor can be time-division multiplexed with that of other sensors, or allocated to a dedicated channel depending on the quantity of data produced (see figure 2), e.g. output from the ECG sensor is in

standard RS232 serial format in packets that begin with ASCII sequence `$SE0` (where S = streaming, E = ECG, 0 = minor sensor number) and carry a payload of 100 bytes, transmitted at 19,200 baud.

Device 2: Blood Glucose Monitoring

We have complemented our work with continuous unobtrusive monitoring by exploring medical devices that rely on self-reporting. We have adapted a system for the monitoring and self-management of Type 1 diabetes to utilise Grid services. The original system was developed in collaboration between Oxford University and e-San Ltd, a mobile healthcare company. The system is built on clinical experience and has evolved based on feedback from both clinicians and patients. A Grid-based version of the system, as demonstrated by this project, offers a possible solution to scaling up the data collection and patient advice in a generic way.

Type 1 diabetes mellitus involves the destruction of beta cells in the pancreas, preventing the body from producing insulin. Without treatment, blood glucose levels will be abnormally high, causing long-term complications such as damage to eyes and nerves. The standard treatment for Type 1 diabetes involves injections of long-acting insulin at bedtime and extra short-acting doses to counteract the effects of each meal during the day. This treatment has been shown to delay and reduce complications, but brings a risk of hypoglycaemia (low blood glucose levels) if insulin doses are too high. Patients must therefore adjust their insulin doses carefully, based on measurements of their current blood glucose levels and taking into account the meals they are about to eat and/or any exercise they intend to do.

In conventional treatment, patients attend a three-monthly clinic to discuss any problems or alterations in treatment with a diabetes specialist clinician. Patients may write down their blood glucose measurements, insulin doses and other information in a paper diary. However, important trends in the data (such as dangerous overnight hypoglycaemia) will not be seen by the clinician for up to three months. The diary itself may not be accurate; since some patients may "adjust" the blood glucose levels they enter in order to appear to have better control.

The e-San system uses an off-the-shelf GPRS (General Packet Radio Service) mobile phone to improve communication between the patient and clinicians. A popular blood glucose meter (the Lifescan SureStep) can be connected to the phone via a specially-built interface cable. When the patient takes a blood glucose reading, it is transferred via the cable to the phone, which immediately sends it to a central server using a GPRS connection. Custom software on the phone prompts the patient to answer a short series of questions about the insulin dose they are about to inject and relevant lifestyle information (diet, exercise, minor illnesses, etc.) The process of gathering this data has been made as rapid as possible for the patient and the system is highly portable: many younger patients already carry a mobile phone and are familiar with the technology. The stored patient data on the server can be viewed both by the clinicians and that particular patient by accessing a secure web page. When required, the clinicians can rapidly contact patients using text messages or voice calls to their phones.

Figure 3: Diabetes self-management system: mobile phone, blood glucose meter and cable

Figure 4: Sample lifestyle questions

The system is currently being used in a 100-patient trial involving young adults with Type 1 diabetes. Feedback from patients and clinicians has influenced the Grid demonstrator version of the system. The principal motivation for developing a Grid version of the phone system is to address the technical challenges of a future scale-up of the system on a national basis. In particular, distributed storage and processing of the data will be required since a single server is not sufficient to handle the data from all Type 1 diabetes patients in the UK (approximately 100,000). It would also be advantageous for a team of diabetes experts to be continuously available to review the data (operating across multiple time zones would allow 24-hour coverage to be achieved with all clinicians still working during normal office hours.) In this application the Grid offers a possible future-proof and generic solution to the scalability problems.

In the following section we consider the development of the underlying architecture used to support these different classes of medical device and the extensions we have made to the Globus Toolkit

to allow this form of device to make itself available across the grid. The infrastructure we have used has also been exploited to undertake remote monitoring via the Grid [11].

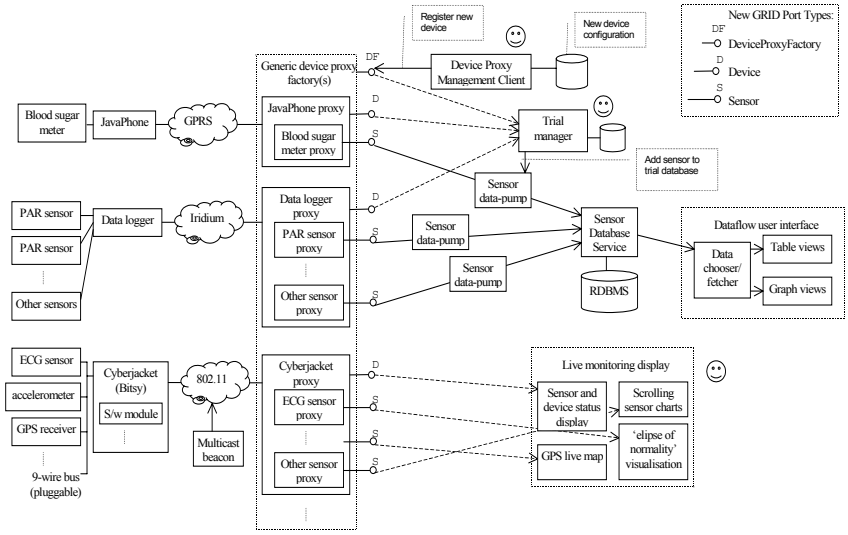

Figure 5: The Equator MIAS devices Grid deployment

Putting Devices and Sensors on the Grid

The devices and sensors comprising our medical devices have considerably less power than those normally expected to be placed on the Grid. They have limited computation power, little memory and low bandwidth, intermittent network connectivity. Such devices are not directly suitable for hosting Grid Services given current technologies. However, we envision certain high-end devices or devices in the near future will be capable of doing so.

Consequently, our approach has been, wherever possible, to make such devices and sensors available as if they were first class Grid Services. We have defined two new application-independent port types: one for a generic sensor, and one for a generic device (which is assumed to host a number of sensors). These port types can be supported directly by sensors and devices of sufficient capability and reliability of communication. However, all of the devices and sensors that we are working with at the moment depend on proxy Grid Services to implement these interfaces on their behalf. The devices communicate with their respective proxies using whatever protocol is appropriate, as and when communication channels are available.

To realise this approach, we have also defined a Generic Device Proxy Factory port type, which can be implemented by factory services able to create device and sensor proxies. We have initially created proxies for the two devices described in this paper and the EQUATOR IRC environment e-Science project's Antarctic lake monitoring device [11].

Of these, the heart monitoring device is the most sophisticated, supporting dynamic addition and removal of sensors as well as the execution of custom software modules on the device itself, e.g. to support on-device monitoring of critical health indicators. Note, even this device cannot support a standard Grid Service interface, since its wearer will often move outside of wireless network range as they go about their everyday activities. In terms of a fixed Grid this would represent a failure of the service (or the Grid fabric). In the context of mobile and remote devices it is a perfectly normal part of operation and should not appear as a failure – the wearable device's Grid Service proxy (on the fixed network) maintains availability based on cached historical data and queues requests for reconfiguration even when the device itself is currently unreachable.

Data Archival

In many applications in health informatics the data from sensors and devices will normally be archived to a persistent store of some form, from which the user (e.g. the clinician or informatician) will request data as and when they require it, e.g. when reviewing a particular case, or testing a new feature detection algorithm.

The second part of our system assumes that devices and sensors are exposed as Grid Services using the port types already mentioned; it then allows the data from these sensors to be archived in a timely fashion to an archival database.

We have designed a custom Grid Service port type for this database, to tailor it to the kinds of data being stored and queries being performed. In the first version this is implemented directly over a Relational DataBase Management System using JDBC; this will migrate to use OGSA-DAI in the next version, in order to provide a generic Grid interface to the data archive.

Data analysis and visualisation

There are two distinct elements of data analysis and visualisation within our current approach:

– Live sensor and device monitoring; and
– Analysis and visualisation of archived data.

The live monitoring component makes direct use of the defined sensor and device Grid Service port types to provide real-time status information. This is appropriate for interactive use, demonstration, trouble-shooting and configuration.

Working with archived data uses the data archive service rather that the sensors and devices themselves, and is more appropriate for routine (occasional) monitoring and longer-term analysis. To support this activity we are developing a simple visual dataflow tool to allow non-programmers to get data from the archive and to view it in various ways. These elements are shown together on the deployment diagram shown in figure 5.

New port types

This section describes the port types that we have defined. We have sought to keep as close as possible to the design idioms of OGSA – as distinct from other distributed object systems such as CORBA or Java RMI – and it may be that others will find these examples illustrative in this respect.

Sensor port type

The sensor port type is in some ways is the most fundamental for the kinds of systems being considered. Each sensor port corresponds to exactly one sensor, which is characterised as:

– some self-describing metadata, e.g. what kind of sensor it is,
– its configuration, e.g. its sampling rate, and
– zero or more data samples (of whatever phenomenon it senses).

As far as Grid applications are concerned we consider the sensor service's Grid Service Handle (GSH) to be the globally unique identity of the sensor (although it typically also exposes other forms of identification that are not specific to the Grid). We expect a sensor Grid service to be persistent, as long as the sensor exists 'on the Grid'. Clearly, this does not prevent the sensor service migrating between hosts or being activated and deactivated by its run-time support environment.

In OGSA, a service exposes information about itself or its internal state using Service Data Elements (SDEs) that can be queried and modified using standard operations in the common Grid Service port type. SDEs, analogous to properties in JavaBeans, support a standard introspection mechanism (again as SDEs) that is used by the standard Java-based service browser GUI provided with GT3, allowing a user to browse the service data of arbitrary Grid services.

Port types can also extend (multiple) existing port types, if they also wish to support those interfaces. In our case, the sensor port type extends the following standard OGSI port types:

– GridService: required by all Grid services to provide standard access to lifecycle operations and service data, and
– NotificationSource: allowing clients to listen asynchronously for changes to the service data.

and defines the service data elements shown in the following tables:

Table 1:Sensor port type: self-description

Name	#	Mutability	Modify?	Description
IdentifiedAs	1	Constant	False	Sensor ID, names and type
Description	1	Mutable	False	Expanded description, e.g. placement, accuracy, etc.
MeasurementTemplate	1	Constant	False	The format in which measurements are reported
MeasurementDiscard-PolicyExtensibility	1..*	Constant	False	Acceptable XML schema types for the measurementDiscardPolicy SDE
MeasurementPublication-PolicyExtensibility	1..*	Constant	False	Acceptable XML Schema types for the measurementPublishingPolicy SDE
ConfigurationExtensibility	1..*	Constant	False	Acceptable XML Schema types for sensor configuration SDE
ProxyStatus	1	Mutable	False	Current status, e.g. in contact with proxy or disconnected

Notes:

– The various …`Extensibility` SDEs are analogous to OGSI's `GridService findServiceDataExtensibility` SDE, and give the names of XML schema types that are understood by a particular Grid Service instance when used in the corresponding SDE (of XML schema type `ogsi:ExtensibilityType`, which is just a wrapper for XML Shema's `any`).
– We are evaluating SensorML [12] as a possible standard XML schema for sensor self-description; in the mean time we are using a simple placeholder type.
– The `measurementTemplate` value allows a potential client to assess the kind of data that will be returned from a sensor even if it has not yet made any measurements. (It is also used internally in configuring the Sensor proxy in our current implementation.)

Table 2:Sensor port type: Externally modifiable configuration

Name	#	Mutability	Modify?	Description
MeasurementDiscard-Policy	1	Mutable	True	The conditions under which the sensor should discard historical measurements
MeasurementPublishing-Policy	1	Mutable	True	The conditions under which the sensor (proxy) should make a new measurement public
configuration	0..*	Mutable	True	Sensor-specific configuration information, e.g. sample rate

Notes:

– We do not expect or require that a sensor should retain its readings indefinitely. The `measurementDiscardPolicy` allows the conditions under which the sensor and/or the proxy discard old readings to be managed. A simple example would be to retain no more than some number of historical readings.
– Different kinds of sensors may take measurements at widely varying rates, and each measurement may represent differing quantities of data. The `measurementPublishingPolicy` allows the way in which the sensor reveals (publishes) new measurements to be controlled. For example, a very rapidly sampling sensor might be configured to announce new measurements no more than 5 times per second (even if it is taking and making available 5000 measurements during this period).

Table 3: Sensor port type: measurement

Name	#	Mutability	Modify?	Description
Measurement	1	Mutable	False	The most recent measurement made by the sensor
MeasurementCounter	1	Mutable	False	A running counter of measurements made
MeasurementHistory	1	Mutable	False	The complete known history of measurements

Notes:

– The `measurement` SDE nominally exposes the most recently made measurement. However the `measurementPublishingPolicy` may mean that some measurements are not actually 'published' as a distinct value of this SDE.
– The `measurementCounter` SDE exposes a running total (monotonically increasing) of the measurements performed by the sensor. Because the `measurementPublishingPolicy` may limit the measurements exposed directly through the `measurement` SDE, a client that requires all measurements should monitor this counter, and obtain any newly available measurements via a query to the `measurementHistory` SDE (below).
– The `measurementHistory` SDE exposes the sensor's full (retained) history of measurements. Depending on the `measurementDiscardPolicy` this may be quite a limited subset of measurements made, e.g. the last few measurements. We also define a new `GridService::findSequenceData` query expression type, `queryByServiceDataNameAndCount`, which (unlike the normal `queryByServiceDataNames` query expression type) allows a client to request only a subset of the available measurement history by providing a measurement counter range of interest. Following standard OGSA idiom, a query expression of the appropriate type is passed to the `GridService::findSequenceData` operation and the service responds accordingly.

Sensor Measurement Types

We have defined `measurement`, `measurementHistory` and `measurementTemplate` SDEs to have type `ogsi:ExtensibilityType`, i.e. any. However, we have also had to choose a default XML schema type for use in the services that we have developed and deployed. We have adopted XSIL, the eXtensible Scientific Interchange Language [13] as a standard XML format for exchanging sensor measurement data. In particular, XSIL defines a simple XML representation of a table of data, with self-defined column names, types and units, each row of which can then represent a single measurement (see sample below):

```
<XSIL>
 <Comment>Temp at depth of 3m</Comment>
 <Param Name="sensorHandle" Value="…handle…"/>
 <Table>
 <Column Name="dateTime" Type="datetime"/>
 <Column Name="counter" Type="int"/>
 <Column Name="reading" Type="float"
Unit="C"/>
 <Stream Delimiter=",">2003-07-07
10:11:12.000000, 123, 1.124,
   2003-07-07 10:11:17.000000, 124,
1.126</Stream>
 </Table>
</XSIL>
```

Notes:

- We have adopted the standard ISO-8601 date-time format, and extended XSIL to support this as a first-class column type ("`datetime`").
- The "counter" column corresponds to the `measurementCounter` SDE's value.
- In some cases (e.g. from the archival database) the sensor's Grid Service handle may appear as an explicit column in the table (if the table may include data from multiple sensors).
- A multi-dimensional sensor, such as a two-axis accelerometer, would have one column for each dimension of measurement (e.g. "X" and "Y" rather than just "reading").

Actuator Port Type

We plan to design a corresponding actuator port type to allow notifications to propagate from the Grid back to sensing devices (e.g. the wearable). Our current focus has been on developing sensing from lightweight devices, however.

Device Port Type

The Device port type is somewhat simpler than the sensor port type. We regard a device as being a supporting or hosting hardware/software platform for zero or more sensors. We assume that each device is characterised by:

- some self-describing metadata, e.g. what kind of device it is, and
- zero or more references to the sensors that it hosts.

As with sensors, we consider the device service's Grid Service Handle (GSH) to be the globally unique identity of the device as far as Grid applications are concerned, and we expect a device Grid service to be persistent, as long as the device exists 'on the Grid'.

The device port type extends the following standard OGSI port types:

- GridService: required by all Grid services to provide standard access to lifecycle operations and service data, and
- NotificationSource: allowing clients to listen asynchronously for changes to service data.
- ServiceGroup: exposing the device's sensors in terms of their sensor Grid services.

Exposing of its sensors through the ServiceGroup port type is the main function of the device service. A ServiceGroup supports notification by default, so a client of the device Grid service can receive notifications of sensor addition and removal.

The device port type also defines the following additional service data elements (table 4). These mirror the corresponding SDEs in the sensor port type.

Generic Device Proxy Factory Port Type

The generic device proxy factory is specific to our proxy-based implementation of device and sensor proxies: a Grid service directly hosted by a sensor or device would have no need of such a port type. We include it here for completeness.

The device proxy factory port type is very simple and extends the following standard OGSI port types:

- GridService: required by all Grid services, to provide standard access to lifecycle operations and service data, and
- NotificationSource: to allow clients to listen asynchronously for changes to service data.
- ServiceGroup: exposing the device proxies that it has created, and used for device discovery.

Factory: allowing clients to request the creation of new device proxy service instances.

Table 4: Device port type: self-description

Name	#	Mutability	Modify?	Description
IdentifiedAs	1	Constant	False	Device ID, names and type
Description	1	Mutable	False	Expanded description, e.g. including placement, accuracy, etc.
ProxyStatus	1	Mutable	False	Current status of sensor, e.g. in contact with proxy or disconnected

It defines one piece of static (port type) service data, which is a value for the `ogsi:createServiceExtensibility` SDE, identifying the specific XML schema type required as the creation parameter to `Factory::createService`. We have also added an additional operation, the Device port type `getCreationInfo`, allowing a client to obtain instance-specific information from newly created device proxy services. This is a work-around for `Factory::createService` not returning the `ExtensibilityOutput` in GT3 release 1.0 (and is an otherwise standard OGSI notification factory).

The particular XML schema required by the generic device proxy factory allows the requesting client to override the normal service deployment properties, e.g. instantiating the correct `GridServiceBase` sub-class appropriate to the device that they are connected to the Grid. The client can also specify an initial set of sensor proxies to be created, as well as provide custom configuration information specific to their device and/or sensor proxies.

At present we provide a simple command-line client to request the creation of a new device proxy. However, some devices (such as the wearable) would be capable of directly requesting the creation

of the Grid proxy, although this would raise additional security considerations.

Database port type

We have defined a custom port type for our sensor data archive, so that most clients can work in terms of domain-specific types (such as XSIL measurement records), rather than having to use a generic database interface and types. Internally, the database service maps to these generic operations.

The database port type is a standard operation-based interface, comparable to a CORBA or RMI interface. It supports addition and querying of: Devices; Sensor types; Sensors; Sensor self-descriptions; and Sensor measurements.

The user interfaces

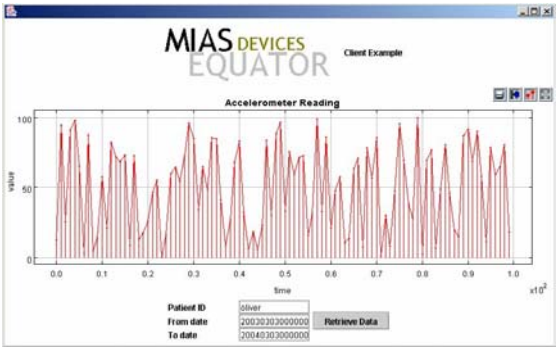

Figure 6: An initial standard interface

The approach we have adopted allows the clinician exploiting the grid infrastructure to view the world as a series of Grid services providing a set of live clinical data. The clinician can view this data and manipulate it using standard data manipulation services. The clinician experience of these devices viewed from the grid is of a set of abstract data services. This data can then be presented using standard grid and web based facilities. Figure 6 is a screenshot of one of our early monitoring interfaces displaying the information arriving from the Grid services associated with the wearable device. We are in the process of refining these interfaces for use by clinicians by undertaking consultation with our clinical partners to establish their requirements.

Summary

In this paper we have provided an overview of our work to date on extending the Globus Grid Toolkit (GT3) to support lightweight mobile devices suitable from medical monitoring. We have presented an overview of the devices we have developed and the highly extensible underlying architecture and services deployed to connect these devices to the Grid. While we hope our approach can act as a blueprint for others seeking to conduct clinical trials on the Grid, before such architecture can be used in real clinical trials a number of significant challenges remain. Of these, a principle issue for future consideration is how we may provide appropriate security and address the privacy demands of this class of application.

Acknowledgements

We would like to acknowledge the EPSRC in supporting our work in the Equator GR/N15986 and Grid based Medical Devices for Everyday Health: GR/R85877 projects.

References

1. Foster, I. and Kesselman, C. (eds.). The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999.
2. The Globus toolkit available from http://www.ggf.org/ogsa-wg/
3. The Global Grid Forum http://www.gridforum.org/
4. The Open Grid Services Architecture (OGSA) http://www.globus.org/ogsa/
5. de Roure, D., Jennings, N., Shadbolt, N "Research Agenda for the Semantic Grid: A future e-Science infrastructure" http://dcs.gla.ac.uk/Nesc/general/technical_papers/DavidDeRoure.etal.SemanticGrid.pdf
6. Open Grid Services Architecture Database Access and Integration (OGSA-DAI), http://dcs.gla.ac.uk/NeSC/general/projects/OGSA_DAI/
7. The oxford centre for e-health http://www.medicine.ox.ac.uk/ndog/tmr/
8. The Biomedical Informatics Group at Nottigham University http://www.eee.nott.ac.uk/medical/
9. Computer Assisted Reporting of Electrocardiograms, Glasgow University http://www.gla.ac.uk/departments/medicalcardiology/research/care.html
10. Muller, H. and C. Randell, "An Event-Driven Sensor Architecture for Low Power Wearables", ICSE 2000, Workshop on Software Engineering for Wearable and Pervasive Computing, pp. 39-41, June, 2000.
11. Greenhalgh et al., eScience from the Antartic to the Grid. To appear proceedings of the 2nd UK eScience All hands meeting.
12. SensorML Specification, http://vast.uah.edu/SensorML/
13. XSIL Specification, http://www.cacr.caltech.edu/projects/xsil