Remote Engineering and Virtual Instrumentation Using Jini Technology

Z. Juhasz, G. Bognar, K. Kuntner, A. Pasztory and Sz. Pota
Pannon University, Department of Information Technology, Veszprem, Hungary

***Abstract*—The rise of Internet and web technology have transformed our society by providing access to unprecedented amount of information, supporting communication and collaboration, empowering e-business and e-commerce. Its role in advancing science is equally important. It has become possible to disseminate scientific information more effectively, access large databases, share computational resources, measurement devices, control experiments and collaborate at a global scale. The human user centered traditional world wide web technology, however, cannot provide the suitable technological platform necessary for the further development and improvement of online and remote engineering applications. New, service-oriented technologies are required, which support higher level of automation, enable seamless programmatic and interactive access to remote devices and systems, and have an adequate programming model suitable for creating large and complex distributed scientific applications. In this paper, we describe such a technology, Jini technology, and illustrate with case studies its various benefits for the remote engineering, virtual instrumentation community.**

***Index Terms* — Remote Engineering, Virtual Instrumentation, Service-oriented Architecture, Jini Technology**

I. Introduction

The rise of the Internet and the World Wide Web opened up new opportunities in teleworking, remote and online engineering. Researchers and developers have been looking at ways to provide access to hardware equipment (sensors, instruments, actuators) in order to improve their utilization or minimize the travel required for their use. Notable examples can be found in the fields of astronomy (remote access to and control of telescopes) and health services (remote diagnostics, pathology, etc).

Due to its relative simplicity, most of these integration efforts are based on web technology and use the web browser as the client program. To provide improved control and interaction, the Java language and platform have been a popular choice for implementation. Applets provide a convenient way of extending the capabilities of web pages with dynamic update and control facilities which are required for these applications [1][2]. The main drawback of the web browser based approach is that it requires user presence. A human user must download the web page and manually control the instruments. This represents serious problems in automated systems operated by programs, e.g. feeding sensor values into a large Grid application for processing and control.

Recent technological advances raised expectations and opened up new possibilities for remote engineering and virtual instrumentation. It is possible to create Internet-based simulations or integrate with Grid systems to enable the development of new scientific applications.

In this paper we show that Jini Technology [3] provides many benefits for virtual instrumentation and remote engineering. It relies on the power of the Java platform, offers programmatic or user interface based access to remote services, and comes with a suite of technologies required to build reliable distributed systems.

The structure of the paper is as follows. Section II discusses requirements that differentiate online and remote engineering as well as virtual instrumentation from traditional web applications. Section III provides a brief introduction to service-orientation and Jini technology. It explains the fundamental concepts, infrastructure, operation and programming abstraction of Jini, and highlights how it supports the requirements listed in Section II. Section IV describes case study applications that illustrate the use of Jini in the online, remote engineering and virtual instrumentation area. The benefits of the Jini model in programming, interaction and flexibility is also discussed. The paper ends with our conclusions and discussion of future work.

II. New Requirements

The Internet is a global computer communication backbone originally designed for a limited set of applications such as FTP, Telnet and electronic mail. The advent of the World Wide Web and the Web Browser opened up the Internet for the public as it simplified the process of publishing information on interlinked web pages without requiring programming knowledge. The ubiquitous browser and its ability to display multimedia content (initially images only) played a crucial role in popularizing this technology.

The web was designed for retrieving mainly static HTML pages. Web servers are therefore stateless, providing the same response for every query regardless the history of interaction with a given user. Online and remote engineering, as well as virtual instrumentation applications are, however, very different from the original web concept. The main differences are due to the following requirements.

- Continuous interaction – Clients need interactive connection to remote systems, devices and instruments as they need to be monitored and controlled continuously.
- Clients need to be able to receive event notifications from the remote system when important state changes occur.

- Various interaction patterns must be supported using a range of data formats. One off instructions, periodic polling, receiving and sending data (ASCII, binary, audio or video) streams to and from the remote systems are equally important.
- Miniaturization resulted in various portable devices (mobile phones, Personal Digital Assistants) that should be supported either as client or remote server devices regardless their resource limitations.
- Fully automated applications in which programs interact with other programs without the intervention of human users are becoming increasingly important. One such example is sensor networks. This type of operation requires an infrastructure and programming model suitable for integration in distributed software systems.
- In automated operation mode, systems must work in the presence of errors. Fault handling, tolerance and minimal administration needs are essential required features.
- Support for composition is important when assembling complex systems from existing modules and devices. This is prerequisite, e.g. for integration into Grid applications.
- Multi-user access to shared devices, data, applications and experiments is essential to enable collaborative engineering work.

The above requirements are not unique to remote engineering applications. Traditional web sites comprising a set of static HTML pages are quickly disappearing. Users increasingly interact with remote *systems* (web applications) instead and consequently expect the desktop application experience. With traditional web technologies this is difficult to achieve.

III. JINI TECHNOLOGY AS A TOOL FOR VIRTUAL INSTRUMENTATION

A. Service Orientation

A relatively new approach to creating complex, typically distributed, systems is service orientation [4]. In a service-oriented architecture, software and hardware are represented as services. Anything can be a service as long as it can be defined by a programmatic interface that describes the functionality offered by the service. The interface serves as a contract between the service user (the client) and the service. Since the implementation of the service is not part of the contract, it is hidden from the client allowing various devices, instruments, etc. to appear on the computer network as virtualized software services.

Service orientation offers a modular, component-based and compositional approach to system design. It specifies the fundamental concepts of service-oriented architecture without prescribing its implementation. Most notable implementations are Web Services Technology [x] and Jini Technology [3]. In this paper, we concentrate on Jini Technology, as we believe it is more suited to and offers more benefits to the online, remote engineering and virtual instrumentation community.

B. Jini Technology Overview

Jini technology is a service-oriented technology based on the Java platform. Its main features are dynamic networking and mobile code. A Jini system consists of services, clients and at least one lookup service that is a mediator connecting clients to services. Jini services are described by Java interfaces creating a contract between the client and service. Service implementations dynamically join the service network by discovering one or more lookup services (registrars) and then registering with them by uploading a Java object that represents the service to the client, as shown in Fig. 1. Clients wishing to use services also discover lookup services using the well-known interface of the required service. The result of the successful lookup operation is the downloading of the service proxy objects from the lookup service to the client, where it will automatically create an access point to the service.

The use of the lookup service makes the system extendable and evolvable in run-time by allowing the addition and removal of services without the need of worrying about hard-wired service addresses. Also, only correctly operating services can be discovered by clients as non-responding services are automatically deleted from the lookup service after a predefined timeout period. This is referred to as the Leasing mechanism that underlies the automatic resource management and robustness of a Jini system.

The use of the proxy object has many benefits. Since the proxy is downloaded from the service, no prior client-side installation is necessary. This provides the simplicity of use similar to applets and ensures that the most recent version is used, thus removing the need of manually upgrading software. The proxy object is a Java object that client programs can readily use, hence services can be easily integrated into other programs and accessed programmatically without human intervention. Jini also provides facilities for event generation to enable services to notify their clients about changes in their state. Transaction support and loosely coupled object-based communication are also supported.

The proxy can carry additional information about the service in form of Java attribute objects. These can describe e.g. the physical location, developer, owner, etc of the service. Attributes enable clients to locate services based on information other than functional properties.

Attributes can be used to attach a user interface object to the proxy as well. If a user interface is required for accessing the service, the Jini ServiceUI mechanism [6] enables services to attach multiple user interface objects to

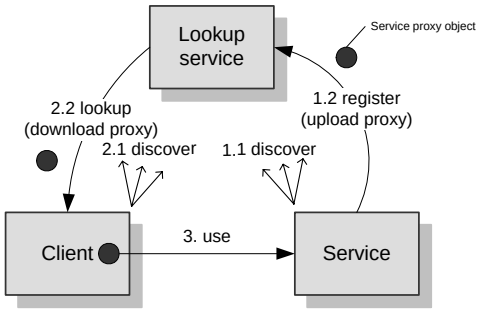

Figure 1. High-level operation of a minimal Jini system

the proxy (various graphical, sound, etc interfaces). The interface is instantiated on-demand based on the requirements of the user.

The final advantage of the proxy concept is that it hides the client-service communication protocol from the client as it is a private implementation detail between the proxy and the service. This allows a service implementer to use the best suited and most efficient protocol for the task or choose any suitable implementation for the service backend.

C. Jini on Limited Resource Devices

The abovementioned scenario assumed participants capable of running the Java Standard Edition version 1.4 virtual machine. In the instrumentation and hardware world, there are many devices that can only run earlier or more limited virtual machines, or cannot run Java at all. Jini can be used in these circumstances as well, although in slightly different ways as we elaborate next.

1) Limited services

The responsibility of a Jini service is to register a proxy object with lookup services and respond to requests received from the proxy. Systems that can run Java 1.3 can participate in a Jini 1.2 system. This is similar to the current Jini 2.1 version but with limited support for security. For this reason, this option is only recommended for closed systems. One such example use of Jini is the security management system of the Eiffel Tower [7].

Services that run 1.0-1.2 versions of Java can use the Jini Surrogate Architecture [8] or the JMatos framework [9] to participate in the service community. The Surrogate Architecture defines a standard operation mechanism in which a limited service can delegate its Jini-related responsibilities to a Surrogate Host – this will represent the service in the Jini world – and perform its service-related tasks without Jini. The service can use any suitable interconnect protocol to connect to the Surrogate Host.

The JMatos framework unifies the lookup service and the particular service in question. The result is a service that can be discovered via its "built-in" lookup service. This removes the need for registration, lease management, etc. The service only has to export the proxy object, which can even be done without Java as the JMatos C version exemplifies this.

If a third party provides a proxy for a given service, it is also possible to implement the service in any programming language as the proxy can connect to the service via any suitable protocol, e.g. raw TCP/IP socket.

2) Limited clients

Clients are more reliant on Java as they need to be able to download and instantiate service proxy objects. Those devices that cannot run Java 1.4 can either use Jini 1.2 (using the Java 1.3 VM) or use the previously mentioned Surrogate Architecture. In the latter case, a purpose-written client adapter is required that connects the client user interface and the proxy object running in the Surrogate Host computer.

An illustration of a Jini 1.2 client is shown in Fig. 5, while a Surrogate Architecture based client is shown in Fig. 6. Both examples are described later in the paper in more detail.

Figure 2. The Jini service browser after discovering the DataSource service

D. Client-side Service Execution

A very important property of Jini Technology is its ability to move executable objects around the network. This property can be exploited in creating virtual instrumentation applications. Since it is the developer's responsibility to decide which part of the system is executed in the proxy object and which part in the remote service implementation, it is possible to create services that execute entirely in the proxy within the client computer. This is equivalent to downloading and installing a virtual laboratory software. In this case, however, the installation happens at run-time and automatically, and the software stays at the client only until the laboratory experiment is performed. This is an ideal framework for creating e.g. educational laboratory exercises.

IV. Demonstration Services and Applications

The main focus of the work of our project team is to look at how Jini Technology could be used for creating reliable, dynamic service-oriented Grid systems. Over the years we have developed the JGrid framework [10] as an experimental device to test our concepts. Although the main emphasis of our work was on computational grids; as part of this effort, we have also looked at how remote instruments and devices could be virtualized and represented as services in a Jini service federation. In the rest of the paper we describe our experience and ongoing work in this area using representative case study application we have developed and explain how they relate to remote engineering and virtual instrumentation.

A. Real-time Data Display – the DataSource Service

Many devices (measurement instruments, sensors) provide continuous real-time data that must be logged, displayed, stored or processed at a different location. To demonstrate this functionality, we have designed a generic DataSource service. It allows clients to continuously read data from services. The actual service is described with a very simple interface. It provides only one method, to retrieve an InputStream object that represents the channel that will carry data to the client.

```
public interface DataSourceService {
    InputStream getDataStream() throws
                RemoteException;
}
```

Figure 2 shows a Jini service browser developed for the JGrid project that already discovered a DataSource service; the service appears in the services list and its visual icon is also displayed. Also shown in the bottom-left window of the browser the user interface (UIDescriptor) and other descriptive attributes attached to the service. Double-clicking the service icon opens up the service interface as shown in right-hand frame of the window in Figure 3.

Should a programmatic client need to use the service, the returned InputStream object is used to read timestamped service data represented as TimeValuePair objects:

```
public interface TimeValuePair {
    public long getTime();
    public double getValue();
}
```

Time stamping is necessary to carry the actual measurement time with the data instead of relying on the client clock.

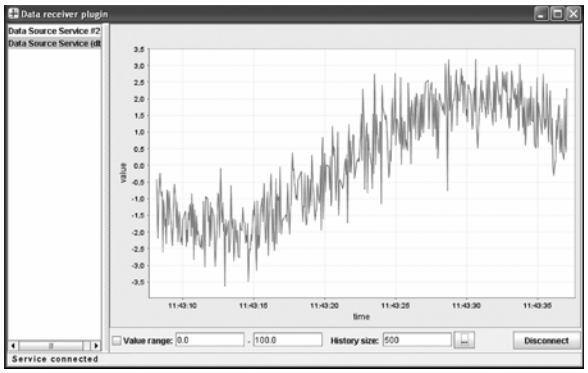

Figure 3. The on-demand graphical user interface of the DataSource service displaying real-time data received from the service.

B. Remote Media Stream Processing

Many applications require support for media streaming. Security surveillance systems, telescopes, microscopes can transmit live video stream for remote viewing and processing. Acoustic sensors may need to transmit live audio data. To illustrate these types of applications, we have developed a remote motion detection system that demonstrates the integration of a live video data source into a grid application for real-time (interactive) processing task. In our experiment, whose configuration setup is shown in Figure 4, a client (located in Hungary) transmitted captured live video stream to a motion detection process running at a remote grid node (located in Australia). The processed, motion detected stream was then transmitted back to the client for display purposes. The experiment successfully demonstrated the capability of a wide-area grid system to support real-time processing of a continuous data stream. Naturally, the data stream could be replaced by other types of data, e.g. audio, binary measurement data, etc. With little modification, it could support e.g. viewing remote microscope images.

Another example application demonstrating media processing is a remote dictating machine, in which a client program was used to capture audio data from a microphone and transmit it as a live audio stream to a

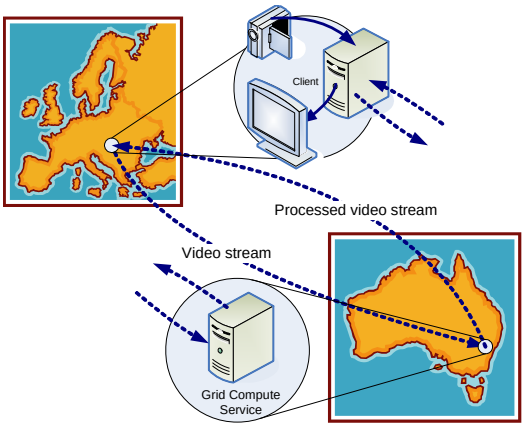

Figure 4. The remote motion detection system that demonstrates live video stream processing at a remote location.

remote dictating machine service. The audio stream was converted and stored on the remote server for later playback and processing purposes. This application demonstrates that acoustic sensors, devices can be integrated into complex applications in a simple manner.

C. Sensor Integration – the Jini Weather Service

The previous examples did not include real hardware devices at the service side. A Weather Service application was developed to demonstrate how hardware can be described and represented as a software service, as well as to provide an example for interfacing between the various system modules.

The sensor used in our example is a 1-Wire Weather Station [11] that measures temperature, wind speed and direction. The measurement probe is connected to a TINI [12] evaluation board that runs a small Java program reading the measurements and transferring it to the Jini Weather Service. The Weather Service runs on a separate host as a full Jini service. In the current implementation the TINI program connects automatically to the service, which will register in the Lookup Service after receiving the first measurement from the hardware. Note that it is possible for the service to detect the TINI module or use the Surrogate Architecture to place the service in the TINI device. Jini gives great freedom for the developer in designing the final system architecture.

The three measured values are represented by three separate interfaces. Temperature is represented as a TemperatureSensor service, wind speed is WindSpeedSensor, while wind direction is given by the

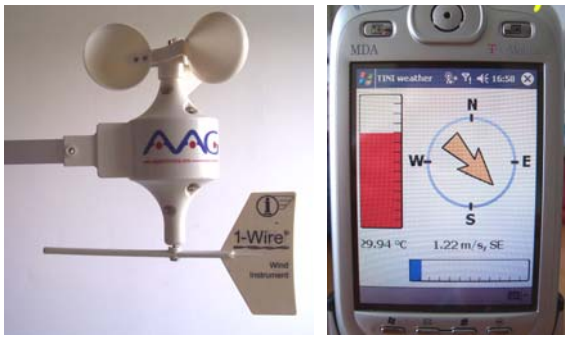

Figure 5. The measurement probe of the Jini Weather Service and its client user interface running on a wireless LAN enabled PDA device.

WindDirectionSensor service. The corresponding service interfaces are shown below.

```
public interface TemperatureSensor {
    public DoubleValuePair getTemperature()
        throws   ServiceCommunicationException,
                    SensorException;
    public double getMinTemperature()
        throws   ServiceCommunicationException,
                    SensorException;
    public double getMaxTemperature()
        throws   ServiceCommunicationException,
                    SensorException;
}

public interface WindSpeedSensor {
    public DoubleValuePair getWindSpeed()
        throws   ServiceCommunicationException,
                    SensorException;
}

public interface WindDirectionSensor {
    public static final String N = "N";
    public static final String NNE = "NNE";
      ...
    public static final String WNW = "WNW";
    public static final String NW = "NW";
    public static final String NNW = "NNW";

    public StringValuePair getWindDirection()
        throws   ServiceCommunicationException,
                    SensorException;
    public double getWindDirectionAngle()
        throws   ServiceCommunicationException,
                    SensorException;
}
```

The service proxy implements all three interfaces thus represents the complete weather station hardware as one entity. It is also possible to create separate proxies for each interface. In this case, a client interested only in temperature would not need to download wind measurement related program code.

The service also contains its own ServiceUI specification-based graphical user interface. Figure 5 shows the interface instantiated on a PDA after discovering the weather station service. The client is in continuous connection with the service and refreshes the screen at a predefined frequency.

The developed application could be used in larger sensor networks, e.g. in meteorological monitoring networks to collect real-time measurement data and, optionally, send it to further services for further processing. The dynamic networking and discovery features of Jini could be used to deal with errors and partial failure.

D. Access from mobile phones

The most complicated development scenario for a Jini environment is having clients that cannot run full Jini client programs. As a case study we have developed a stock value monitoring system that can be easily generalized for accessing any service providing real-time or near real-time data. The client runs a specific Java MIDlet that connects to a Surrogate Host executing the Jini client that can access the Jini Stock Service.

Figure 6 shows two screenshots of the client program (here running in an emulator). The left screen allows the user to select which stocks he or she is interested in, while the right screen shows the graphical display of the retrieved stock values. It is easy to see how this framework can support the monitoring of other types of services via a mobile phone.

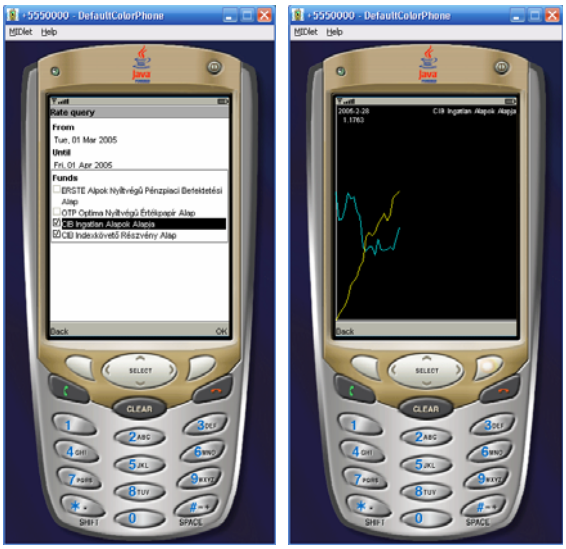

Figure 6. The Jini service browser after discovering the DataSource service

E. Remote Control to Services

All case studies mentioned so far read values from services or transferred data to them but did not control services. Remote control facilities are equally important in remote engineering applications. We have deliberately left this requirement to the end as it naturally emerges from the service-oriented nature of the system. Each method on a service's interface can be viewed as a remote control command to the service. The role of the developer is to develop an interface that correctly and effectively describes the remote control capabilities of the remote device. As an example, we show a hypothetical basic microscope interface below. This allows a client to position the microscope lens above the specimen, set magnification and retrieve the live video stream. The complete interface depends on the capabilities of the real microscope in question.

```
public interface Microscope {
    public RTPStream getImageStream()
          throws RemoteException;
    public void moveTrayLeft()
          throws RemoteException;
    public void moveTrayRight()
          throws RemoteException;
    public void moveTrayUp()
          throws RemoteException;
    public void moveTrayDown()
          throws RemoteException;
    public void setMagnification(double value)
          throws RemoteException;
}
```

V. CONCLUSIONS

Online and remote engineering, virtual instrumentation are becoming increasingly important fields in science and engineering. The increasing cost of high-tech instruments and the collaboration needs of small and medium enterprises with academia lead to systems that allow effective access to and share of instruments.

In this paper we have argued that traditional web technology is not suitable for achieving these goals. A service-oriented approach is required that represent remote applications and devices as services.

Jini is a very elegant and powerful technology for creating dynamic service-oriented systems. We believe that its object-oriented programming model, support for fault-tolerance, ability to work with and without human intervention, its architectural flexibility offers many advantages in virtual instrumentation and remote engineering.

We demonstrated using a number of case study applications that it can be used in various application domains for reading, processing and displaying live data of various sorts, controlling remote systems or creating virtual laboratory programs usable at and on-demand basis creating a rich user experience superior to currently available alternative technologies.

In the near future, we are planning to design and develop new types of services (including e.g. actuators) to demonstrate further application possibilities, and in the long term we are hoping to use these ideas and expertise for solving real remote engineering and virtual instrumentation problems.

REFERENCES

[1] iPath - Open Source Telemedicine Platform: http://ipath.sourceforge.net/

[2] I. Ahmed, H. Wong and V. Kapila, "Internet-Based Remote Control using a Microcontroller and an Embedded Ethernet Board", http://www.parallax.com/dl/docs/article/EEBnBS2Paper.pdf

[3] J. Waldo, K. Arnold, *The Jini Specifications. Jini Technology Services*, Addison-Wesley, Reading, MA, USA, second edition, 2001.

[4] P. Allen, *Service Orientation*, Cambridge University Press, 2006.

[5] The World Wide Web Consortium, Web Services Activity, http://www.w3.org/2002/ws/

[6] B. Venners, "The ServiceUI API Specification", http://www.artima.com/jini/serviceui/Spec.html

[7] Nedap Security Management, Nedap AEOS, http://www.nedap-aeos.com/en/solutions/

[8] Jini Technology Surrogate Architecture Specification, surrogate.jini.org/sa.pdf

[9] PsiNaptic Corp., JMatos, A Jini technology for embedded processors, http://www.psinaptic.com

[10] The JGrid project: http://pds.irt.vein.hu/jgrid

[11] AAG Electronica. LLC, 1-Wire Weather Instrument Kit, http://www.aagelectronica.com/aag/index.html

[12] Dallas Semiconductor, TINI platform, http://www.maxim-ic.com/TINIplatform.cfm

AUTHORS

Z. Juhasz and his co-authors are with the Department of Information Technology, Pannon University, Veszprem, 8200, Hungary (e-mail: juhasz@irt.vein.hu).

Manuscript received June 14, 2006. This work was supported by the Hungarian National Office for Research and Technology under Grant GVOP-3.1.1-2004-05-0035/3.0 and GVOP-3.1.1-2004-05-0266/3.0