Messaging solution for a serial hardware device

Asked 16 years, 3 months ago Modified 12 years, 6 months ago

Viewed 628 times

1

I have a serial hardware device that I'd like to share with multiple applications, that may reside on different machines within or spanning multiple networks. A key requirement is that the system must support bi-directional communication, such that clients/serial device can exist behind firewalls and/or on different networks and still talk to each other (send and receive) through a central server. Another requirement of the system is that the clients must be able to determine if the gateway/serial device is offline/online.

This serial device is capable of receiving and sending packets to a wireless network. The software that communicates with the serial device is written in Java, and I'd like to keep it a 100% Java solution, if possible.

I am currently looking at XMPP, using Jive software's Openfire server and Smack API. With this solution, packets coming off the serial device are delivered to clients via XMPP. Similarly, any client application may send packets to the serial device, via the Smack API. Packets are just byte arrays, and limited is size to around 100 bytes, so they can be converted to hex strings and

sent as text in the body of a message. The system should be tolerant of the clients/serial device going offline, meaning they will automatically reconnect when they are available again, but packets will be discarded if the client is offline. The packets must be sent and received in near real-time, so offline delivery is not desired. Security should be provided by messaging system and provided client API.

I'd like to hear of other possible solutions. I thought of using JMS but it seems a bit too heavyweight and I'm not sure it will support the requirement of knowing if clients and/or the gateway/serial device is offline.

java hardware messaging xmpp

Share

Improve this question

Follow

edited Feb 3, 2009 at 15:19

asked Sep 18, 2008 at 18:39

Andrew

177 ● 1 ● 1 ● 9

3 Answers

Sorted by: Highest score (default)

1

Jini might fit the job. It works really well in distributed environments where multicast is available but it also works in unicast, and is quite fast. Not only it provides remote services, but also remote events, and distributed transactions if you need them. A downside is that it only works with Java.

Where I work, Jini is used in a infrastructure with more that 1000 machines, with each machine providing remote services used to access the devices connect to the machine serial ports.

Share Improve this answer

Follow

answered Oct 3, 2008 at 10:25

jassuncao

4,777 ● 3 ● 31 ● 35

1

You really need to provide a bit more detail... do the clients need guaranteed delivery? What about offline delivery? Is this part of a larger system? Do you need encryption? Security?

If you want the smallest footprint possible, then should transmit data using SocketServer, Sockets, and serialization. But then you lose all of the advantages of the 3rd party solutions you mentioned, which typically include reliability, delivery guarantees, security, management, etc.

I would personally use JMS, but that's because I'm familiar with it. There are a number of stand-alone servers

that can be deployed out-of-the-box with virtually no configuration. They all provide for guaranteed delivery, some security, encryption, and a number of other easy-to-use features. Coding a JMS publisher or subscriber is pretty easy.

Update: If you want the most ease in coding, then I would look at the third-party solutions. Looking at Smack/XMPP, the API seems to be a little easier than a JMS for the functionality you asked for. You still have to setup/configure a server, etc.

The Smack API also has a lot of extra baggage that you don't need either, but its "concepts" are a little more intuitive since its all chat/IM concepts.

I would still look at OpenJMS or ActiveMQ. I think knowing JMS will be more valuable in the future as compared to knowing XMPP. Take a look at their Getting Started documentation or the Sun Tutorial to see how much coding is involved. In JMS parlance, you will want an administered "Topic" and a "Queue" where the Serial Port App will receive and send messages respectively. All of your clients will open a subscription to the Topic and send their outbound messages to the Queue. When you send messages, their delivery mode should be non-persistent.

edited Oct 7, 2008 at 14:36

answered Sep 19, 2008 at 15:45

James Schek

18k ● 7 ● 53 ● 64

0

I ended up using XMPP via the Smack API. What led me to this decision was its native support for presence (is the client online/offline) and robust connection handling (it automatically reconnects if a the underlying connection breaks). Another benefit of XMPP is that it's compatible with Google Talk, so I don't need to setup a server. Thanks for the suggestions. In case anyone is interested, I have released the code on Google Code http://code.google.com/p/xbee-xmpp/

Share Improve this answer

Follow

answered Feb 3, 2009 at 19:36

Andrew

177 ● 1 ● 1 ● 9