www.psinaptic.com

PsiNaptic Inc.
1410, 10655 Southport Rd. SW
Calgary, Alberta
Canada T2W 4Y1
T: 403 720-2531
F: 403 720-2537

Illustrations and Descriptions of JMatos™ and CMatos in Vehicles and Devices

Introduction

The concept of code mobility and how devices can offer information intrinsic to itself as a "service" to another device is not an easy concept to understand. Purpose of this document is to illustrate and describe, in detail, two scenarios in which devices are able to interact with each other, offering and/or consuming Java-based services as a JavaObject. The JavaObject can contain anything from a simple driver to a full-blown application, complete with a graphical user interface (GUI). The two exemplary scenarios are:

1) Vehicle or mobile device is able to offer services intrinsic to itself. In the case of a vehicle, the services can be diagnostic, location, front or rear LCD screens, Hands Free or audio to another device. In the case of a mobile device, the services can be a user profile/preferences, navigation or MP3 files. (CMatos software on the vehicle or mobile device)
2) Vehicle or mobile device is able to both offer and consume a service. In this case, in addition to the vehicle or mobile device being able to offer services intrinsic to itself, these devices can also consume a service such as games, video and loyalty brand programs. (JMatos software on the vehicle or mobile device, capable of dynamic class loading)

Framework Description

The framework under discussion for JMatos and CMatos software is Jini™, a specification for distributed computing produced by Sun Microsystems™. This framework is based on the concept of code mobility in which, as devices require services or applications, the code is obtained from a LookUp Service (LUS) and executed on the device. The code itself can implement the service in any manner it chooses. For example, the service can use a client/server model, a Remote Procedure Call (RPC) model, look for further Web Services to fulfil its contract, or simply execute the service on the device that requested the service.

The code mobility is based on the Java concept of code being byte code that is able to execute on any Java Virtual Machine (JVM). This means that the 'Jini client', or service consumer, must be capable of executing Java byte code. This does not mean that the LUS must be Java however.

For a comparison of Jini to other technologies, http://www.javapassion.com/jini/JiniAndOtherTechnologies4.pdf is a good place to start.

How it Works

With Java/Jini the attributes and behavior of a device can be discovered and downloaded as Java byte code and instantiated by another device. In the case of a Jini enabled MP3 player or mobile phone entering the vehicle offering their services, the vehicle can discover these services, download and instantiate the service on the vehicle. The vehicle now knows everything it needs to know about these devices and can control the device from the vehicle control panel.

Here's briefly how it works:

- When a device (or vehicle in this case) using Jini enters a network and has services to offer, it will advertise these services through its LookUp Service (LUS). It will use a multicast announcement to advertise its presence. Any device in a network that is looking for services will locate these LUSs and be able to search for the services it is looking for. When it finds the ones it wants, it will acquire the service (which can be a driver, GUI or application) from the LUS and run it locally on its JVM. The applications will then perform the intended functions.
- Whenever a service or device leaves the network or is no longer available, the service object, applications or drivers are automatically removed from the memory of the device that was using the service, leaving the memory available for other functions. A leasing mechanism is used to accomplish this. Jini also provides event notification for added flexibility in communications between applications.

PsiNaptic Software Products – JMatos and CMatos - Description

JMatos and CMatos is an implementation of the Jini LookUp Service and related protocols that enable very small embedded processors to offer Java based services. One of the unique qualities of PsiNaptic software JMatos and CMatos technology is code mobility which extends the Java programming model to the network by moving data and executables as a JavaObject over the network.

Traditionally, in order for devices to interact with each other, each device needs to have pre-installed make, model, and revision level software. This is becoming increasingly complex, costly and restrictive. **Code mobility eliminates the need for pre-installed software on each device, because devices are able to autonomously and securely offer or exchange information. Aside from reducing intricacy of device interaction, devices can be multi-functional and future proof with automatic updates.**

Another unique quality is JMatos and CMatos very small size. JMatos less than 100kb footprint Jini Service enables devices to spontaneously *offer* or *consume* services on a network. JMatos implements the ServiceRegistrar as a Java Object and is currently running on a number of platforms including Windows, Linux, Win CE, Java 1.1.8 – 1.4, PJava, J2ME (CDC, CLDC), QNX, and OSGi. It has been demonstrated on a Dallas Semiconductor TINI (Tini Network Interface), iPAQ, Zaurus, Xeva and Nokia 9210 running Symbian OS. The small size enables a device to have its own LookUp Service and be fully autonomous.

CMatos, is functionally equivalent to JMatos, but for non-Java devices. Very small-embedded processors can offer Java-based services that can be anything from a simple driver to a full-blown application complete with a GUI. With a total system memory footprint of less than 60 kilobytes (not including offered services), CMatos, complements JMatos. It has been demonstrated on a Dallas Semiconductor TINI (Tini Network Interface) and integrated into Cambridge Silicon Radio (CSR) BlueCore2, (Bluetooth module)

CMatos-BlueCore2 utilizes the standard and ubiquitous TCP/IP network stack (PAN profile) for communication, significantly simplifying application development for Bluetooth devices. In addition, because CMatos-BlueCore2 can offer an API implementation as Java byte code, the implementation can be downloaded and executed by a Jini client when needed. This simplifies application development for Bluetooth devices and future proofs applications that are no-longer dependent on profile specifications. Additional information and Product Sheets can be found at www.psinaptic.com

What to keep in mind when studying the two scenarios:

1- Vehicle makers who decide to put Java/Jini in the vehicle can build an ecosystem for the Java community to build applications; be the first to establish a controlled access proxy to the developer community and monetize the platform; have a more capable consumer platform by being an external interface provider exposing the vehicle interface to mobile devices; use Java/Jini to future proof the platform and enable secure over the air provisioning...
2- In terms of "C" vs. Java, there are significant benefits for using Java/Jini particularly in a multi-tenant environment: 30-50% productivity gains on the development platform due to modularization, scalability, priority execution and life cycle management, as well as amortizing the code over the same processor.
3- There is less of a value proposition for Java/Jini if the vehicle platform is single purpose, built to cost, whereby the vendor has no responsibility to keep it current, or an upgrade is a hardware replacement vs. a software upgrade. On the other hand, if the vehicle platform is to be network capable, be able to execute secure 3rd party code, expose specific elements to mobile devices (API predefined interface), free up profiles in the Bluetooth module... then a Java/Jini solution set is worth considering.
4- The chicken and eggs scenario: Vehicle makers cannot offer a vehicle based feature or technology that is contingent on code change on mobile devices. This is true for any technology – that is why there are standard groups and pre-agreements on both sides etc. What is so compelling about J/CMatos is that it reduces the need for agreements or control over what others do or

don't, because with Java/Jini, the definitions are at the application level (API's), which are much simpler and faster to define and implement.

1- Scenario Description: CMatos on the Vehicle or Mobile Device

Vehicle is able to offer services intrinsic to itself, such as diagnostic, location, front or rear LCD screen, WiFi or Bluetooth Hands Free functionality or access to the vehicle audio system -

Or mobile device is able to offer services intrinsic to itself such as a user profile/preferences, navigation or MP3 files – to another device (Jini Client capable with JVM and Dynamic Class Loading).

Automakers hesitant to add Java to the vehicle, can still enable their vehicles to offer services to another device able to discover and utilize those services.

Advantages:

- **Defining vehicle interfaces at an API (as opposed to Bluetooth profile)**
- **Secure**
- **Cost effective**
- **Robust**
- **Future proofing, automatic updates**

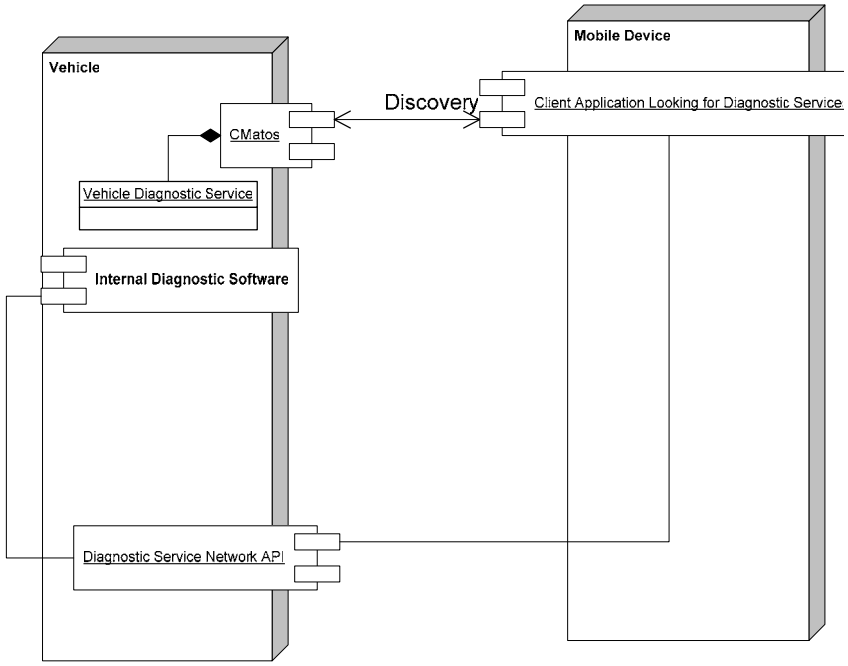


Figure 1 Deployment Diagram

The scenario is composed of 2 devices:

Device one is the **'Vehicle' offering Diagnostic Service**. The Vehicle is assumed to have TCP/IP network capabilities through the Bluetooth (BT) Personal Area Network (PAN) profile or through some other means. The Vehicle is NOT capable of dynamic class loading (DCL), and is therefore not capable of downloading a Jini/Java service. This does not prevent the Vehicle from participating in a Jini distributed computing application. The Vehicle in this case will be the entity providing the service with a "Return to Sender" Jini service. i.e. The service knows how to contact the device that supplied the service. The communication between the devices during the Jini discovery portion and code exchange must be via TCP/IP but once the service is on the Jini client the communication is provided by the service it self and does not necessarily have to use TCP/IP i.e. it could use BT RFCOMM to talk back to the vehicle.

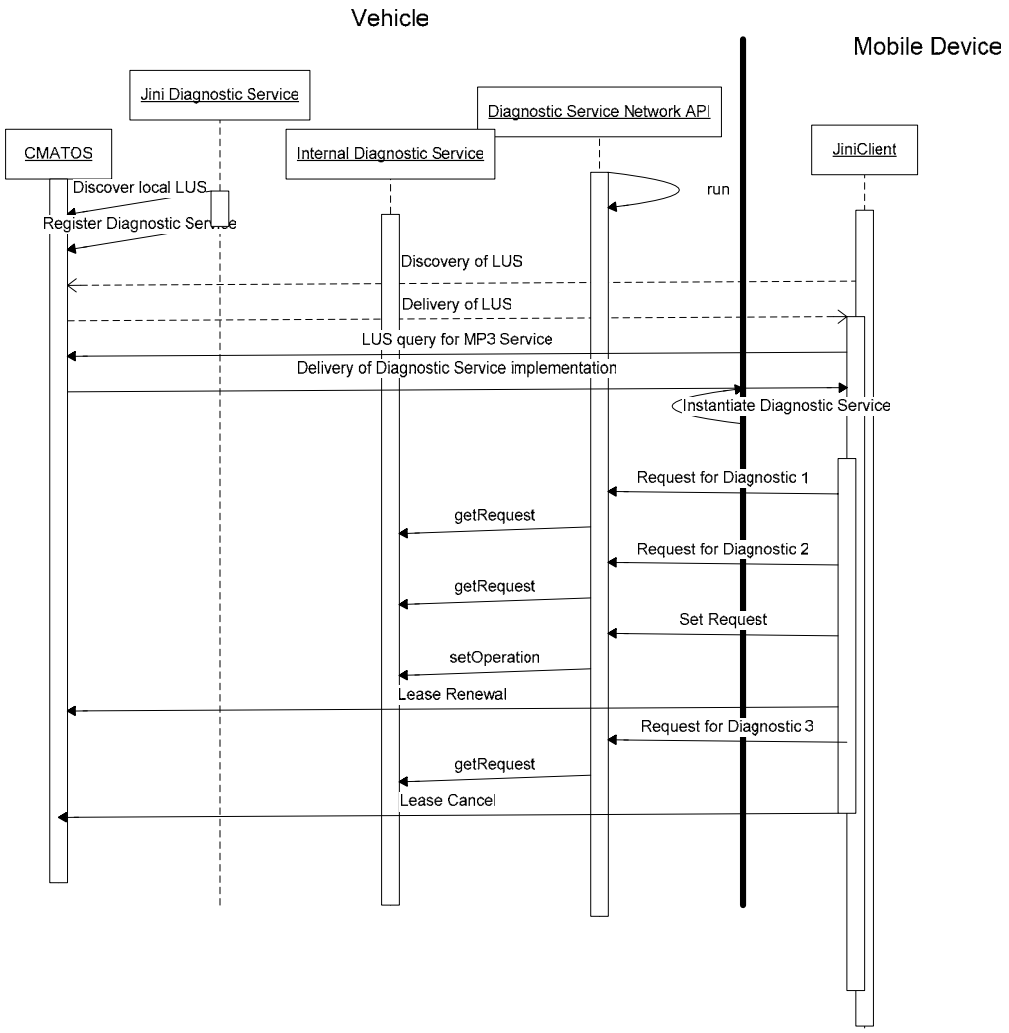
The second device is a Jini client capable device: **'Mobile Device'**. This means that the device is capable of un-marshalling an object, is network capable, and is capable of dynamic class loading through URL class loading or some kind of URLHandler. This is at a minimum a J2ME Connected Device Configuration (CDC) with the Personal Foundation (PF) and higher type device such as a vehicle.

The Jini discovery protocol will allow the devices to discovery each other and allow the Mobile Device to 'Lookup' a Java Diagnostic Service (JDS) (or Hands Free Service, or front/rear LCD Service…) on the vehicle over TCP/IP.

Once the Mobile Device has determined that there is a LUS within network range, and that this LUS has the JDS the Mobile Device is looking for, the Mobile Device then requests the LUS to deliver the implementation of the JDS to the Mobile Device through a Jini retrieval request.

The CMatos LUS on the vehicle delivers the marshaled object (in byte code) to the Mobile Device. The Mobile Device un-marshals the implementation object as provided by the LUS and using the code base in the marshaled object, loads the class byte code to the Mobile Device. The JDS is now an instantiated object on the Mobile Device and knows how to use the JDS Network API as offered by the vehicle to obtain things like engine or transmission diagnostic information etc. from the vehicle. The Mobile Device does not know HOW this is implemented, simply that it now has an API that it can get the information it wants. The implementation knows how to talk back to the vehicle and get the information.

A more detailed view is shown in the following sequence diagram.


Vehicle
Mobile Device
Jini Diagnostic Service
Diagnostic Service Network API
CMATOS
Internal Diagnostic Service
JiniClient
run
Discover local LUS
Register Diagnostic Service
Discovery of LUS
Delivery of LUS
LUS query for MP3 Service
Delivery of Diagnostic Service implementation
Instantiate Diagnostic Service
Request for Diagnostic 1
getRequest
Request for Diagnostic 2
getRequest
Set Request
setOperation
Lease Renewal
Request for Diagnostic 3
getRequest
Lease Cancel

2- Scenario Description: JMatos on the Vehicle or Mobile Device

Vehicle or mobile device is able to both offer and consume a service. In this case, in addition to the vehicle or mobile device being able to offer services intrinsic to itself, these devices can also consume a service such as games, video and loyalty brand programs. (JMatos on the vehicle or mobile device, capable of dynamic class loading)

In addition to the advantages in the first scenario, devices are able to consume a service, becoming far more multifunctional. In the case of a vehicle, it can now instantiate a user profiles offered by a mobile device, personalizing the vehicle to the users preferences; or potentially download games, videos, maps or highway information from a service provider.

In the case of a mobile device, it can accept a Point of Sale (POS) branded User Interface (UI). A POS machine can offer "its branded UI" to a user's mobile device, reenforcing a fuel vendor or grocery store branding/loyalty programs. The user does NOT have to scroll on his/her device network provider for the vendor. The vendor is interacting directly with the user, positively impacting the user experience and increasing opportunities for micro commerce.

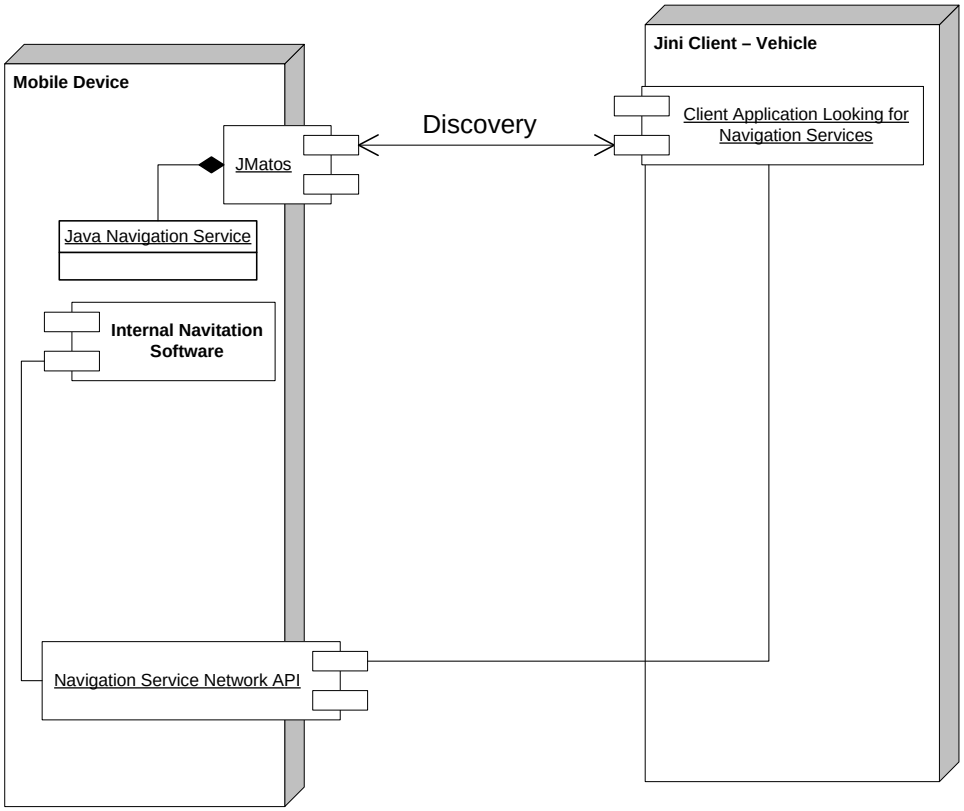


Figure 2 Deployment Diagram

The scenario is composed of 2 devices:

The **Mobile Device** is a Java 2 Micro Edition / Connected Device Configuration (J2ME/CDC) capable device offering **Java Navigation Service**. The Mobile Device is assumed to have TCP/IP network capabilities through the Bluetooth PAN profile or through some other means. In this scenario, the Mobile Device is capable of dynamic class loading, and is therefore also capable of downloading a Jini/Java service (more on this below). For the purposes of this scenario, the Mobile Device will be the entity providing the service with a “Return to Sender” Jini service. i.e. The service knows how to contact the device that supplied the service.

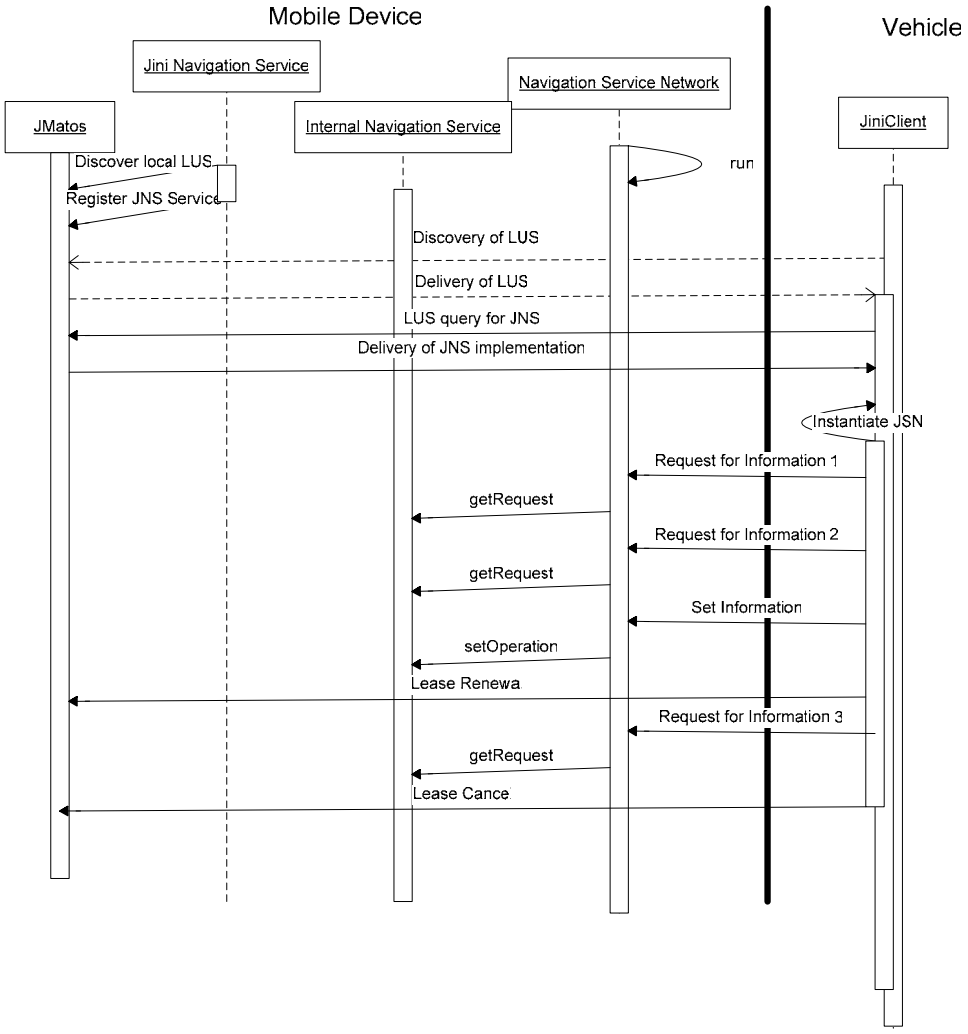
The communication between the devices during the Jini discovery portion and code exchange must be via TCP/IP but once the service is on the Jini client the communication is provided by the service it self and does not necessarily have to use TCP/IP i.e. it could use BT RFCOMM to talk back to the Mobile Device.

The second device is a Jini client capable device, in this case, a vehicle. This means that the device is capable of un-marshalling an object, is network capable, and is capable of dynamic class loading through URL class loading or some kind of URLHandler. This is usually J2ME CDC with the Personal Foundation (PF) and higher type device such as a vehicle.

The Jini discovery protocol will allow the devices to discovery each other and allow the vehicle to ‘Lookup’ the Java Navigation Service (JNS) on the Mobile Device over TCP/IP. Once the vehicle has determined that there is a LUS within network range, and that this LUS has the JNS the vehicle is looking for, the vehicle then requests the LUS to deliver the implementation of the JNS to the vehicle through a Jini retrieval request. The JMatos LUS on the Mobile Device delivers the marshaled object (in byte code) to the vehicle. The vehicle un-marshals the implementation object as provided by the LUS and using the code base in the marshaled object, loads the class byte code to the vehicle. The JNS service is now an instantiated object on the vehicle and knows how to use the Navigation Service Network API (NSN) as offered by the Mobile Device to obtain things like current location, waypoints, routes, etc. from the Mobile Device.

The vehicle does not know HOW this is implemented, simply that it now has an API that it can get the information it wants. The implementation knows how to talk back to the Mobile Device and get the information.

A more detailed view is shown in the following sequence diagram.


Mobile Device
Vehicle
Jini Navigation Service
Navigation Service Network
JMatos
Internal Navigation Service
JiniClient
run
Discover local LUS
Register JNS Service
Discovery of LUS
Delivery of LUS
LUS query for JNS
Delivery of JNS implementation
Instantiate JSN
Request for Information 1
getRequest
Request for Information 2
getRequest
Set Information
setOperation
Lease Renewa
Request for Information 3
getRequest
Lease Cance

Code Mobility at a Glance

Both the vehicle and the device need to have pre-install make, model, and revision level software. Costly, limited, restrictive. No code mobility, no future proofing, no automatic updates.
With J/CMatos, a VM and Dynamic Class Loading (DCL): Vehicle offers services i.e. audio, hands free, LCD screen, diagnostic, location Device consumes services Device sends data to vehicle Vehicle access & displays data **Cost effective, robust, code mobility, future proofing, automatic updates**
With J/CMatos, a VM and Dynamic Class Loading (DCL): Device offers services to the vehicle. The vehicle can instantiate the service and interact with the device. **Cost effective, robust, code mobility, future proofing, automatic updates**
Both vehicle and device can offer and consume a service **Best overall!**

(Connectivity: Wired or wireless)

Vehicle **Device**	No JVM No J/CMatos	With JVM No DCL No J/CMatos	With JVM With DCL No J/CMatos (client)	No JVM With CMatos	With JVM With JMatos No DCL	With JVM With JMatos With DCL
No VM **No J/CMatos**						
With VM **No DCL**						

No J/CMatos						
With VM **With DCL** **No J/CMatos (Client)**						
No VM **With CMatos**						
With VM **With JMatos** **No DCL**						
With VM **With JMatos** **With DCL**						