Jini™ in Formula One
The Field Experience
Two Years Later

Fabrizio Giudici, Tidalwave
Cristian Colonello, Sun Microsystems

The customer

MAGNETI MARELLI

Motorsport Department

Magneti Marelli
Holding
Headquarters
Corbetta (Milan)
Italy

SALES	45 million Euro
EMPLOYEES	120
R & D (of Sales)	13.7 %
BRANCHES (n° 4)	Corbetta (Milan)
	Venaria Reale (Turin)
	Nanterre (France)
	Detroit (USA)

Business areas

SALES BREAKDOWN

F1
70%

Rally
9%

Other
13%

Bike
8%

Partner of '92/'05 champions

	F1 Drivers	F1 Constructors	Rally	Raid	SBK
1992	MANSELL	WILLIAMS-RENAULT	LANCIA	CITROEN	DUCATI
1993	PROST	WILLIAMS-RENAULT		CITROEN	DUCATI
1994		WILLIAMS-RENAULT		CITROEN	DUCATI
1995	SCHUMACHER	BENETTON-RENAULT		CITROEN	DUCATI
1996	HILL	WILLIAMS-RENAULT		CITROEN	DUCATI
1997	VILLENEUVE	WILLIAMS-RENAULT	SEAT		
1998			SEAT		DUCATI
1999		FERRARI			DUCATI
2000	SCHUMACHER	FERRARI	PEUGEOT		DUCATI
2001	SCHUMACHER	FERRARI	PEUGEOT		DUCATI
2002	SCHUMACHER	FERRARI	PEUGEOT		DUCATI
2003	SCHUMACHER	FERRARI			DUCATI
2004	SCHUMACHER	FERRARI			
2005	ALONSO	RENAULT			

Functional requirements

- Realtime data flow distribution
- Publish & retrieve “environmental” information
- Provide facilities for management and monitoring
- Be accessible by generic Java apps
- Integrate with existing MM software with minimal impact

Non functional requirements

- Up to six data flows (3 cars, 2 flows per car)
- Up to 100 clients:
  - Existing MM client applications
  - Customer tools based on JRTTS API
- High availability
  - Switch PBE in a about one second
  - No interruption of service for publishing service
  - Be quick: a car finishes a lap in about 1,5 minutes!

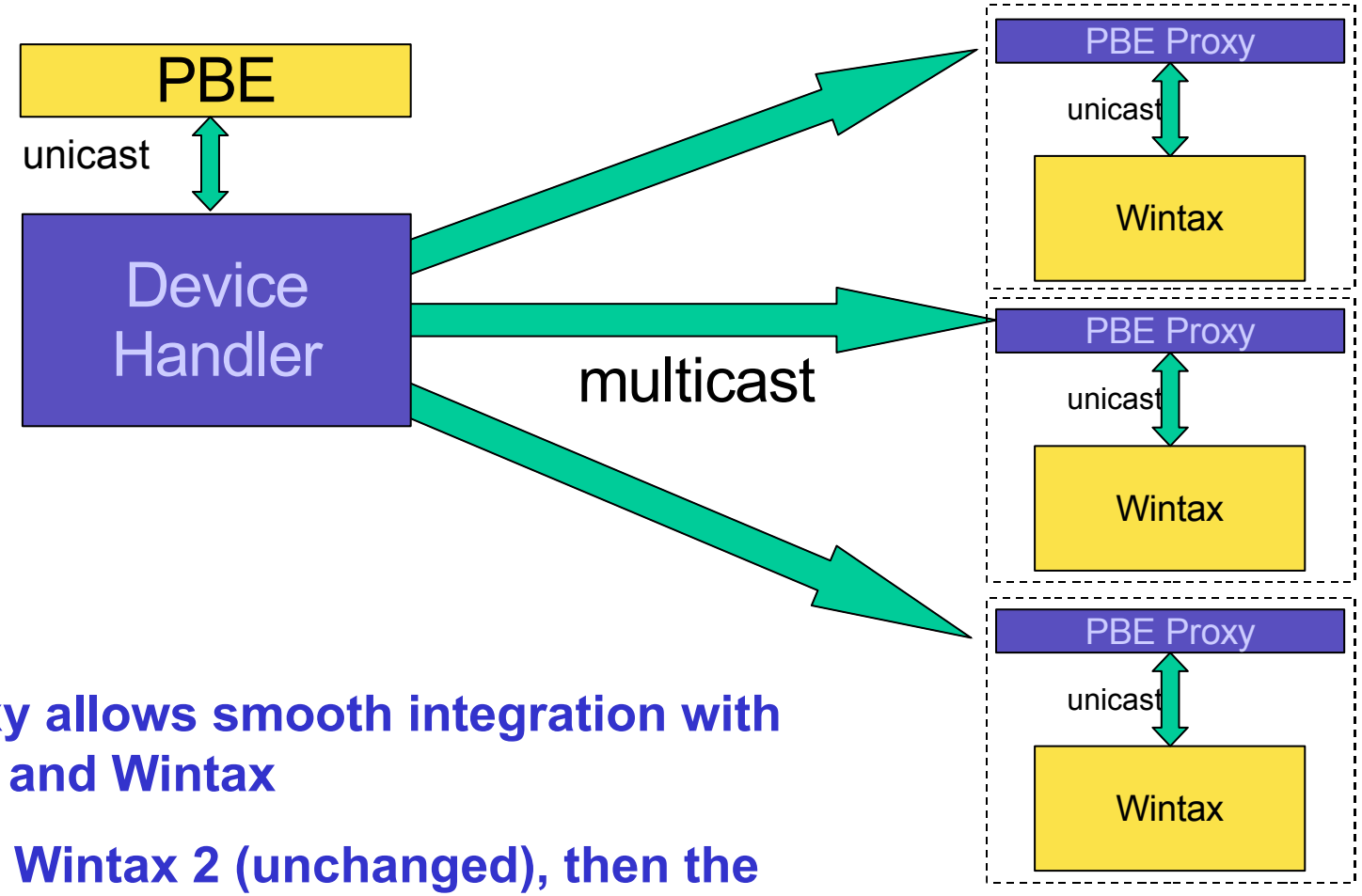
Non functional requirements

- Manageability
  - Fast fault detection
- Robustness
  - Faulty or disconnected cables
  - Hubs erroneously powered off
  - Laser link on the pitwall
- Portability
  - Some teams use Windows, others use Linux

Technologies

- Jini
  - Zero configuration required
- Rio
  - Dynamic provisioning
  - Centralized configuration
- JGroups
  - Flexibility

Scalability by multicasting



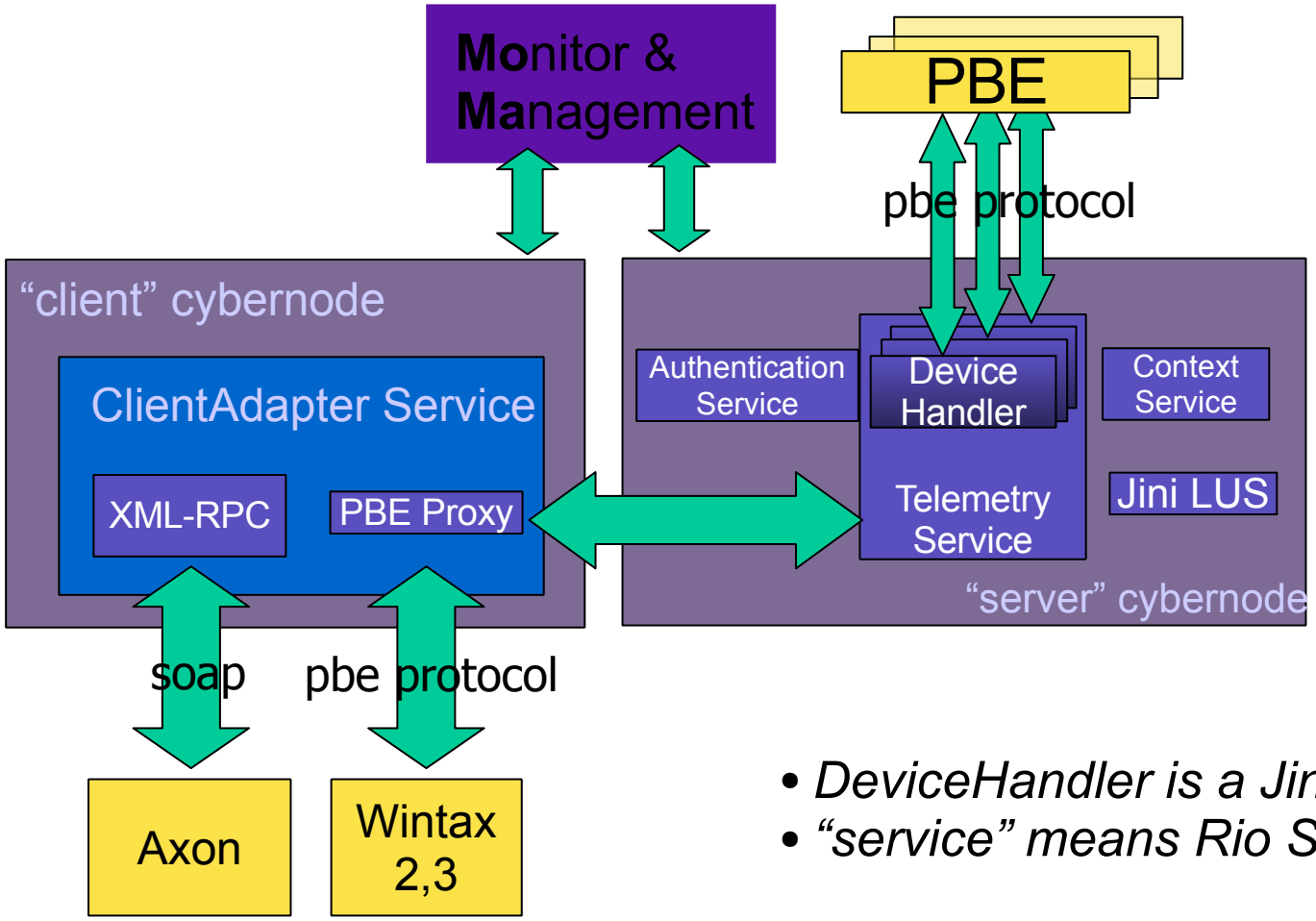
Proxy allows smooth integration with PBE and Wintax

First Wintax 2 (unchanged), then the new Wintax 3

Not only multicast, indeed

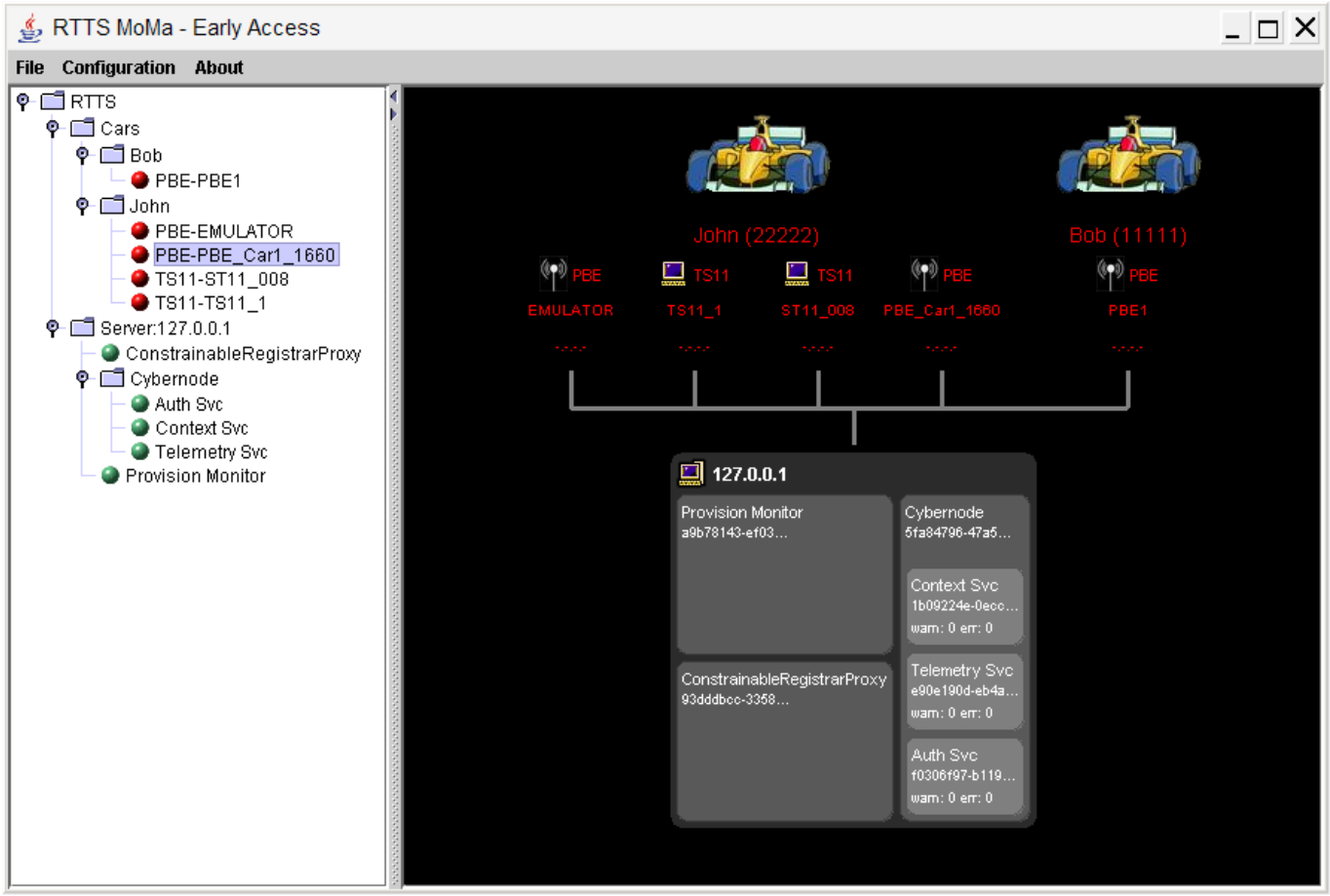
- Plain, unreliable multicast
- Reliable Multicast (frozen)
- TCP/IP
  - A few Critical clients on the pitwall laser link

Basic architecture



- *DeviceHandler is a Jini surrogate*
- *"service" means Rio ServiceBean*

The management tool (MoMa)



Challenges

- Zero configuration
  - And I really mean zero: not even DNS
  - Multiple NIC support
- Unreliable network (cable disconnections)
  - TCP/IP often problematic
- High Availability

Challenges (RFCs, bugs)

- Again problems with the cables
- Work through a router / take control on the TTL

The cable problem

- The “Multicast Announcement” Interval
  - 2 Minutes
  - It was not designed for much shorter times
- Configurable, but setting it too low overloads the system
- We need a heartbeat, at high frequency,...
- ... and something that manages it

What kind of heartbeat?

- Rio provides support for TCP/IP heartbeat
  - Just needs to be configured
- But TCP/IP is too slow in detecting disconnections
  - Also because Java can't fully configure it
  - O.s. tuning not an option (zero config, remember?)

What kind of heartbeat?

- Multicast “Service Heartbeat”
  - Means that a remote ServiceBean is reachable
  - Each Cybernode fires it, with service ids of hosted beans
- Custom component added in each Cybernode by means of standard start-XXX.config
- Interval of 500ms

Service Discovery Extension

- **RTTSServiceDiscovery**, a façade hiding service discovery stuff
- Provides “disconnection” semantics:

```
public interface RTTSServiceDiscoveryListener
 {
  void serviceAdded (ServiceItem serviceItem);
  void serviceRemoved (ServiceItem serviceItem);
  void serviceReachable (ServiceItem serviceItem);
  void serviceUnreachable (ServiceItem serviceItem);
 }
```

Fast Service Discovery

- What if:
  - A client C1 is disconnected (e.g. cable unplugged);
  - A new service C2 is published;
  - C1 is connected again (e.g. cable plugged back)
  - It's likely that C2 publishing event reaches C1 at the next Multicast Announcement, too late

Fast Service Discovery

- A inner **FastServiceDiscovery** fixes this
- Detects reconnection of nodes with a LUS
- Performs explicit LUS lookup and fires events
- **RTTSServiceDiscovery** filters out duplicate events

The Multi-NIC Issue

- Customers have (more than) two distinct subnets on clients
  - Only one is the correct route to the server
- Jini binds server sockets to the “default” NIC
  - Especially on Windows this is unpredictable
  - If Jini chooses the wrong NIC, can't set up TCP/IP “callback” connections from the server!

The Multi-NIC Issue

- Conceptually simpler, but required some work
- Extensive configuration of everything such as:

```
import NetworkInterfaceSelector;
hostAddress =
   NetworkInterfaceSelector.getSelectedHostAddress();
serverExporter = new BasicJeriExporter(
  TcpServerEndpoint.getInstance(hostAddress, 0),
  new BasicILFactory(), false, true);
```

The Multi-NIC issue

- **getSelectedHostAddress()** blocks until it knows the right IP
  - We should prevent Jini from booting if we don't know it
  - Rather brutal, but the alternative was too complex
  - In any case, clients are useless without a connection
- Naïve solutions failed
  - A first try was based on the Multicast Announcement
  - “Just recv from both NICs, only the good one recvs it”

Multicast stacks, and Windows

- Indeed you recv the MA from both NICs! :-(
- We were forced to explicitly send the info
- Another heartbeat: “Node heartbeat”
  - Each Cybernode multicasts its IP and netmask
  - Clients compare and match with IPs bound to local NICs

The Cable, again!

- Discovered a problem with event delivery
- Rio dispatches events serially
- If one of the receivers is disconnected...
- ... we experience TCP/IP timeout delays
- Again, TCP/IP is too slow for our needs

ParallelDispatchEventHandler

- Extends **DispatchEventHandler**, 100 lines

```
public void fire() {
        for (int i = 0; i < resources.length; i++) {
                final ServiceResource sr = resources[i];
                PoolableThread thread =
                    (PoolableThread)threadPool.get();
                thread.execute(new Runnable() {
                    public void run() {
                        fire(event, sr);
                    }
                });
        }
 }
```

“ContextService not avail”

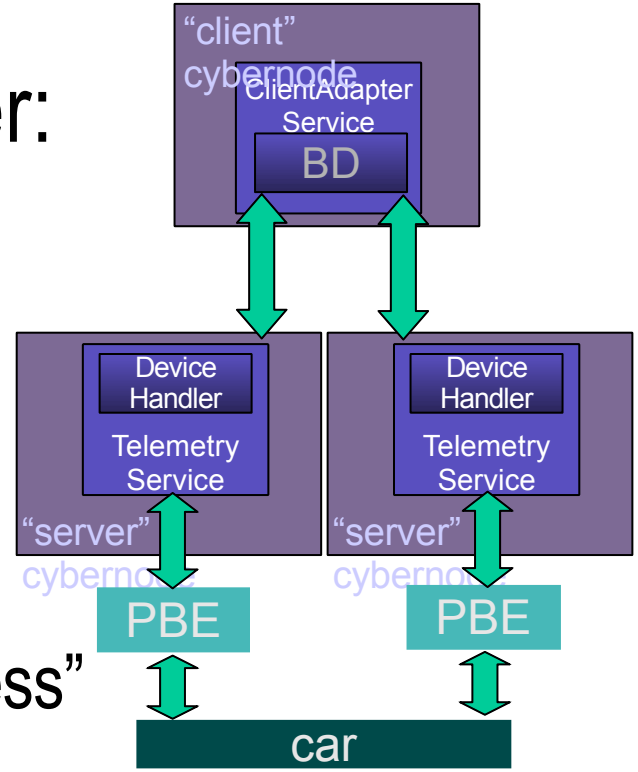
- Occurred seldom
- But required restart of the involved clients
- Probably the most annoying bug
- Due to a bug in **RTTSServiceDiscovery**
  - When the ServiceHeartbeat gets in before the Jini event
  - Prevented serviceReachable() / serviceAdded() to fire
- Lesson learned: always think async, don't assume the simpler sequence of events

Routers and TTL

- RFC: work through a router
- Paranoid control of Time To Live for multicast
  - Not only for the data flows (obvious)
  - Also for the Jini Multicast Announcement / Discovery
- Solution: **NetworkInterfaceSelector.getTTL()**
- Configuration sent in the Node Heartbeat

High availability

- Both for realtime data (easier: stateless) and context data (stateful)
- Implemented on the client
  - Business Delegate idiom
  - Multicast heartbeat to test “liveness”



Lessons learned

- A good team is the key to success
- Use patterns (design by patterns!)
- Test as much as you can in the target production environment, i.e. in the “pit box”
- UP/Sun iterative and incremental, architecture-centric design methodology works

Summing up

- Cable disconnection and fast recovery were the source of 90% of the problems
- Very stressing scenario for Jini / Rio
- Addressed with customizations
  - Jini and Rio are *really* customizable
  - The less TCP/IP, the better
- A Sun Microsystems success story!
  - ***Jini / Rio are WINNERS***

Future work

- Work through a firewall
- Some RFC are not Jini related
  - e.g. the “deferred data”
  - Improvements in the circuit coverage (confidential!)
- Personally, I'd like
  - To refactor the heartbeat stuff
  - Try reliable multicast for delivering events
    - Would simplify design in some parts

Thanks

- MM staff
  - Tom Hyder, Alessandro Maresca, Vaifro Dariol
- The original RTTS team
  - Antonella Balduzzi, Rafal Kowalski, Fabio Romano, Marco Castigliego, Antonio Terreno
- Dennis Reedy! And the Jini Engineering
- Miguel Vidal

Fabrizio.Giudici@tidalwave.it
Cristian.Colonello@sun.com