Real-time Agent Middleware Experiments on Java-based Processors towards Distributed Satellite Systems

Christopher P. Bridges and Tanya Vladimirova

Surrey Space Centre, Faculty of Electronic Engineering, University of Surrey,
Guildford, Surrey, United Kingdom
GU2 7XH
{C.P.Bridges / T.Vladimirova}@surrey.ac.uk

Abstract—Distributed satellite systems are large research topics, spanning many fields such as communications, networking schemes, high performance computing, and distributed operations.[1][2] DARPA's F6 fractionated spacecraft mission is a prime example, culminating in the launch of technology demonstration satellites for autonomous and rapidly configurable satellite architectures. Recent developments at Surrey Space Centre have included the development of a Java enabled system-on-a-chip solution towards running homogenous agents and middleware software configurations.

Modern commercial agent middleware solutions are typically written in Java which is unsuited to real-time mission critical embedded systems due to problems with large standard libraries, a slow and undeterminable execution model, and dynamic class loading times; to name a few reasons. To overcome these issues, an investigation into real-time Java processing & execution technologies has explored methods of implementing hardware-based Java runtime environment (JRE) or Java virtual machine (JVM) for embedded systems.

This paper discusses the key experimental parameters, such the agent application footprint and performance as well as the stack call depth and memory profiling of these threaded behaviours on differing Java execution platforms. A standard agent middleware application which implements IP-based networking was used for experiments towards delay tolerant operations. Additional benchmark applications on many Java-based platforms were also ran to estimate the stabilities and real-time capabilities on emulated and real embedded hardware.

TABLE OF CONTENTS

[1] 978-1-4244-7351-9/11/$26.00 ©2011 IEEE.
[2] IEEEAC paper #1639, Version 1, Updated October 26, 2010

1. BACKGROUND

In terrestrial systems, more recent key technologies include *distributed computing* used in intranets to distribute computationally intensive tasks and wireless network technologies such as those found in laptops for the Internet. Distributed computing is typically enabled by *middleware*, a software layer offering services to connect software components across a network for integration or sharing computing resources. Java based technologies have been key in enabling many distributed applications using the internet protocol suite (IP) and existing application programming interfaces (APIs). In space systems, like other terrestrial systems, the use of IP has become prevalent and has been used on numerous missions [1] [2] [3] for providing point to point based IP communication.

Terrestrial distributed computing is applied to two very different areas. One area aims at highly networked, *high performance computing* (HPC) for virtual organizations with extremely large pools of resources. The other targets embedded environments, such as *wireless sensor networks* (WSN), where computing ability, memory, power and communications are extremely limited compared to high performance computing systems. New paradigms have been used to realize a distributed computing environment using *intelligent agents*, where an agent is a mobile software entity to provide distributed communications or distributed control. Agents can encapsulate any distributed computing paradigm or communication model to perform tasks or set behaviours with no restriction to either data or network centric solutions. To date, terrestrial distributed computing systems use Java for TCP/IP based networking applications and modern Agent technologies use a Java environment which provides services for control/communication applications [4]. Embedded solutions for networked distributed computing software are highly motivated to utilize less memory for reducing overall power consumption.

1.1. Java-based Agent Systems

Modern agent middleware platforms are Java Agent DEvelopment (JADE) [5] or Foundation for Intelligent Physical Agents – Open Source (FIPA-OS) [6], both providing sets of middleware services for agent management, messaging, and mobility. JADE is also

extended for wireless embedded devices: Light Extensible Agent Platform (LEAP). Agents operate on these platforms using behaviours and any Java based APIs as applications on top of the middleware.

Agent middleware solutions are typically written in Java except MANTA [7] which was written in C++ (and has since been dropped). Java is unsuited to real-time mission-critical embedded systems due to problems with large standard libraries, a slow and undeterminable execution model, and dynamic class loading times; to name a few reasons. To overcome these issues, an investigation into real-time Java processing & execution technologies has explored methods of implementing a Java runtime environment (JRE) or Java virtual machine (JVM) for embedded systems.

Table 1: Real Time Java Execution Technologies

Layer	Examples
Safety/Real-Time Specifications	RTSJ [8] & SCJ [9]
Just-in-Time (JIT) Compilation	Cacao [10]
Ahead-of-Time (AOT) Compilation	GCJ [11]
Hardware Implementations of JVM	JOP [12] & SHAP [13]

Table 1 displays some of the main methods for real-time Java execution, from the application layer with specifications or libraries for coding, such as RTSJ [8], to just-in-time or ahead-of-time compilers like Cacao [10] which uses 1 MB RAM, or finally with hardware implementations such as Java Optimized Processor (JOP) [12] or Secure Hardware Agent Platform (SHAP) [13] which are both implemented as soft cores for use in field programmable gate arrays (FPGAs) for *system-on-a-chip* (SoC) solutions. The majority of these Java processing techniques are towards soft real-time guarantees except for hardware implementations, which are aimed at hard real-time guarantees. One of these methods is usually followed by fault-tolerant or real-time requirements. Additionally, all exceptions or errors must be autonomously handled which is not currently achieved in modern middleware solutions and must be addressed.

2. DISTRIBUTED SATELLITE SYSTEM DRIVERS

All space systems are sized based on a specific payload or mission application which is then optimized for low mass to reduce the cost of launch. They typically operate on low power to meet stringent power requirements and need to ensure reliability against the space environment. However, *distributed space systems* (DSS) have many technological drivers such as intersatellite links (ISL) and networking for greater ground communications or computer resource sharing for performing formation flying or clustering missions leading to greater science return per dollar ($). Mobility support from existing efforts can drive new techniques and technologies for greater fault resilient/tolerant operations. All these drivers are integrated into computing requirements for differing mobile ad-hoc network (MANET) scenarios:

- *Formation Flying:* The tracking and maintenance of a desired relative separation, orientation or position between or among spacecraft requires many sensors, a robust propulsion system, and potential computing for potential collisions.

- *Sensor Network:* A loosely coupled group of satellites for obtaining larger temporal or spatial data sets. The QB50 mission is an example of such a mission using 2 kg nanosatellites [22].

- *Virtual Satellite:* Also called Spacecraft Fractionation, a spatially distributed group of satellites working as a single unit to perform a specific mission using intersatellite links will also require robust communications and resource sharing capabilities. Examples include an array configuration to provide a large aperture or DARPA's F6 mission to demonstrate resource sharing, intersatellite connectivity, and cluster operations [23].

The cost of developing space systems suitable for the harsh space environment, launch and ground based control station maintenance often runs into the 10's to 100's of millions of dollars ($). Despite this, the cost of building satellites has reduced through two significant trends; namely, the use of commercial-off-the-shelf (COTS) parts and miniaturisation. In turn, new space mission architectures have been developed such as DSSs. Table 2 summarizes the existing DSS missions and compares the key features: the use of ISLs, the use of on-board distributed computing, the constellation design, and the cost.

Each constellation has some progress towards a complete distributed satellite system but no current mission meets all

Table 2: Overview of Current Distributed Satellite Systems

Mission	Type	Distributed Satellite System
IRIDIUM [14]	Communications	Uses ISLs, distributed telecoms routing, stable constellation, high cost
Emerald [15]	Educational (Tech Demo.)	No ISLs, on-board distributed computing, never launched
Cluster [16]	Science (Magnetosphere)	No ISLs, no on-board distributed computing, varied constellation periods
TechSat-21 [17]	Science (Tech Demo.)	Uses ISLs, distributed formation control, never launched, high cost
Milstar [18]	Communications	Uses ISLs, distributed routing, stable constellation, high cost (military)
NASA ST-5 [19]	Science (Solar Weather)	No ISLs, stable constellation, no on-board distributed computing
FORMOSAT-3/COSMIC [20]	Science (Space Weather)	No ISLs, stable constellation, no on-board distributed computing
PRISMA [21]	Science (Formation Flying)	Uses ISLs, stable relative position, no on-board distributed computing

the requirements towards intersatellite communications with multiple satellites, on-board processing, and the utilisation of smaller cost effective platforms. These DSS missions have been huge missions with high deployment and orbit maintenance costs. The current commercial and scientific missions typically do not attempt:

- Intersatellite connectivity due to the high fuel/propellant cost to maintain a constellation.
- Or on-board processing for data aggregation and collection for autonomous constellation management.

3. Existing Computing Solutions

There are many differing computing platforms that are currently used in space. The PRISMA mission, which has demonstrated intersatellite connectivity in a formation with optimized trajectories, is implemented a little more classically using the MATLAB Real Time Workshop Embedded Coder to convert MATLAB or Simulink code to C++. This is run using RTEMS operating system on a LEON2 architecture (LEON3 for flight), shown in Figure 1 below [24]:

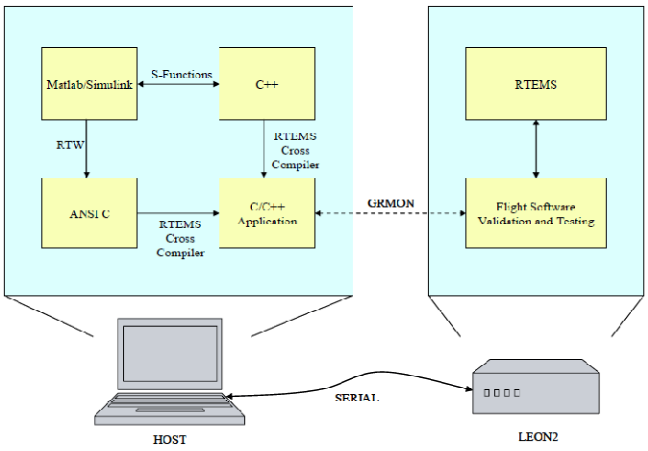

Figure 1: Schematic of the PRISMA navigation software development environment [24]

Using this method, all node behaviours can be modeled accurately in simulation before they are executed on flight hardware.

Previous developments at Surrey Space Centre have focused system-on-a-chip cores for distributed satellite systems [25]. These include a wireless transceiver based on IEEE 802.11 (WiFi) with DMA access and a Java processor, JOP, integrated together with the LEON3 and AHB2 bus scheme. Software developed included the fault tolerant agent middleware called JADE-FT. All these developments are following the trend at implementing towards IP on satellites as uniform and mature techniques for implementing hardware and software systems.

The previous agent middleware solution implemented JADE-LEAP together with a set of additional behaviours specifically for managing multiple instances of middleware for distributed computing applications across an IP-based intersatellite link.

3.1. Target Computing Platforms

For this study, there are three hardware development platforms that can implement Java under test. One is the aJile kits, in particular the aJ-102sk [26], which operate the aJile network direct execution Java processor, aJ-102. The next is a Spartan-3 1500 FPGA board which implements the combined LEON3 and JOP processors [27]. Finally, the next is the Nexus-One smart-phone from Google [28]. These are shown in Figure 2.

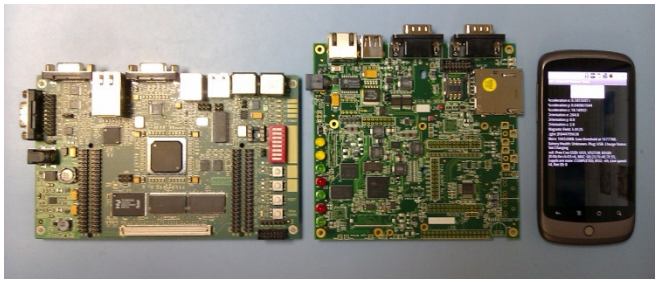

Figure 2: Experimental Hardware

These devices were chosen as they represent a spectrum of state-of-art electronics that could be targeted for space or satellite applications in the near future. The LEON3 FPGA configuration has much heritage, as adopted by the European Space Agency (ESA). The aJile was chosen as another representative 'Java-enabled' hardware platform, this time implemented as an ASIC. The Nexus-One smart-phone was chosen as it has the advantages of a large open-source library and community of native C and Java developers.

Table 3: Comparison of Experimental Hardware

Platform	FPGA	ASIC	Smart-phone
Processor	LEON3 / JOP	aJ-102	SnapDragon
Speed (MHz)	50	165	1000
Internal Mem.	Configurable	32 kB D&I	Unknown
External Memory	8 MB Flash 64 MB SDRAM	32 MB Flash 32 MB SDRAM	512 MB Flash 4 GB SD Card
Interfaces	Ethernet, JTAG, USB, RS232, GPIO	I^2C, Ethernet, JTAG, USB (ZigBee)	µUSB, Bluetooth, WiFi

4. Software Experiments

Previous software tests investigated the application footprint, heap usage, and execution times to complete a standard agent middleware application for differing JADE and FIPA-OS flavors with the aim of operating the final executable on JOP [27]. These experiments however will investigate the alternatives to JOP on the market, including AOT compilers.

The JADE-FT application is used as a common testbench to compile for, with a Java target revision of Connected Device Configuration (CDC) 1.0. It implements previously discussed JADE-LEAP Instance Manager to handle networking failures/outages for networked satellite formations and also software exceptions for single event effects commonly found in Earth orbit.

Embedded software metrics can be taken from both *static* and *dynamic* measurement points. The static includes executable footprint with resultant functionality from included libraries whilst the active measurements can be taken from the RAM usage, timing, and resultant scalability all under test. Fault-tolerance, typically confirmed using environmental tests such as thermal vacuum (TVAC) or total ionic radiation dosing, were not performed.

4.1. Static Measurements of Agent Middleware

The previous comparison of Java solutions for an IP-based distributed computing platform is extended and can be found in Table 4.

Table 4: Executable Memory Footprint Comparison [3]

OSI Software Layer Method	Size (MB)
1. CORBA C++ (LEON3, RTEMS, ORB, 802.11 Driver, TCP/IP Lib.) [29]	1.739
2. Standard Java (LEON3, RTEMS, JRE 1.4 Lib., CDC 1.0, JADE-LEAP)	17.782
3. JOP Compilation (LEON3 & JOP, CDC 1.0, JADE-LEAP)	1.106
3b. JOP Compilation & ProGuard Minimization (LEON3 & JOP, CDC 1.0, JADE-LEAP)	0.305
4. GCJ & GCJ Builder in Eclipse (O3, G0 Options) (LEON3, RTEMS, GCJ Lib., JADE-LEAP)	6.220
5. Fiji-Virtual Machine 0.9.0 (LEON3, RTEMS, Fiji-VM Lib., JADE-LEAP)	1.625
6. aJile 102 Starter Kit (aJile-102, JADE-LEAP)	2.134
7. Nexus-One (SnapDragon, Linux 2.6, Android, JADE-LEAP App)	0.514

It can be seen that the third option offers the smallest memory footprint whilst retaining real-time functionality using a hardware/software design with CDC 1.0, JADE-LEAP, and agent applications at a footprint of 1.1 MB. Compared to a CORBA-based implementation, the proposed system would reduce the footprint by 37 % and this is much higher for the other Java-based ahead of time solutions. From this comparison, it is clear that there are many Java virtual machine ports to the SPARC /LEON3 for integrating real-time Java-based agent applications to embedded platforms using RTEMS or µC-Linux operating systems. Many of the results from Table 4 are developed in Kubuntu Linux using a combination of Eclipse tools, makefiles, and terminal scripts. These are freely available from the multiverse using *apt-get* commands.

The use of minimization tools, such as ProGuard, has shrunk, optimized, and obfuscated JADE-FT extensively on java/jar files. *Shrinking* removes unused classes, fields, and methods. *Optimisation* removes debug and logging code and ensures classes are static and final. *Obfuscation* renames classes, fields, and methods with simpler characters and values. For a fully functioning executable of the JADE-LEAP middleware, communication protocols, management services, and instance management, the Java runtime environment library classes were reduced from 2028 to 115 (94% reduction) and application classes were reduced from 839 to 482 (56% reduction). Even though this results in a reduction in static memory usage, the potential for dynamic classloading applications and *mobile code* methods (a key advantage and argument for agent technologies) is impeded. Applications requiring adaptability require the full functionality without platform minimization.

4.2. Dynamic Measurements of Agent Middleware

As many of the executables were compiled for specific targets, emulated targets are used to make measurements and assess timing and memory usage. For the LEON3, Gaisler's tools such as TSIM can be used and for the Nexus One, the Android Software Developer Kit emulator can be used. The compiled JADE-LEAP app is shown in Figure 3.

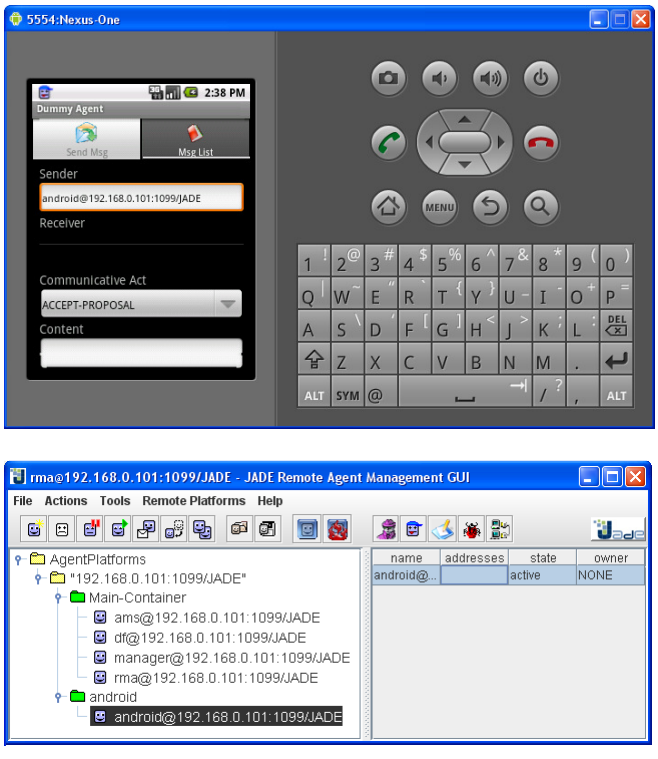

Figure 3: JADE-LEAP & Laptop GUI-Container

A key scenario is when an ad-hoc network consisting of mobile nodes performs topology reconfiguration – where a

[3] Kaffe Virtual Machine 1.1.9 is also available under Kubuntu sources and can be compiled as: LEON3, Linux 2.6, Kaffe VM Lib., JADE-LEAP

new master 'sink' node is assigned. A method probe was used to find out the heap usage and subsequent overhead of disconnecting and reconnecting middleware instances and performing soft resets of the middleware, shown in Figure 4.

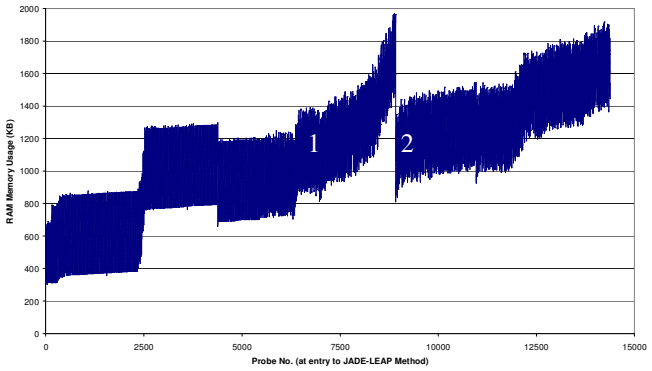

Figure 4: Instance Manager Thread performing Soft Resets on a Laptop target

Figure 4 shows reconfiguration as the main node with 1 additional node and a reconfiguration as a backup node with 2 additional nodes. Three middleware in-stances are connected using some key classes: the runtime instance, properties assignments, and profile implementations. These key classes contain methods which generate hash maps and arrays for holding information on the location and registered Agents at a cost of approximately 200 kB per Agent platform plus an original 600 kB for the first instance. Scalability is a key issue here and as the number of networked nodes increases by 1, the memory consumption also increases which is shown in Point 1 of Figure 4. Upon reconfiguration however at Point 2, the instance is destroyed and restarted under new conditions, in this case, as a backup node where messaging and control is not so centralized. From Point 2, it is also observed that double the methods are called for one more additional networked middleware instance as the mobile phone is discovered and added.

4.3. Java Benchmark Execution Comparison

Due to the absence of true embedded Java benchmarks, the online community of JOP has developed a number of micro benchmarks to measure JVM performance; evaluating the number of clock cycles for single or sets of bytecodes [30]. These include:

- Single & short sequence of bytecode tests such as iinc, iload3, getstatic, and invoke.
- Sieve – Calculates all prime numbers in an incremental expanded array.
- Kfl – An industrial application based on distributed motor control.
-
- UDP Client/ Server – A network application which loopbacks messages.
- Lift – An automated lift controller in a factory.

All applications are self adjusting and stop when the benchmark exceeds 1 second. Of particular interest is the UDP/IP client application which is a highly practical case for cache and stack size evaluation in the Java execution processors. Iteration results were obtained from running these on various implementations can be found in Figures 5-8 for CDC1.0, MIDP, Fiji-VM, and GCJ compilations.

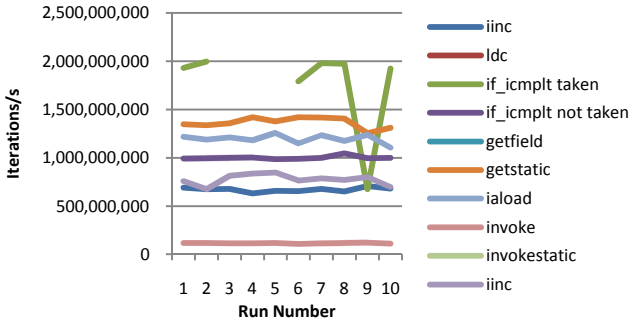

Figure 5: CDC1.0 on Laptop (runnable jar)

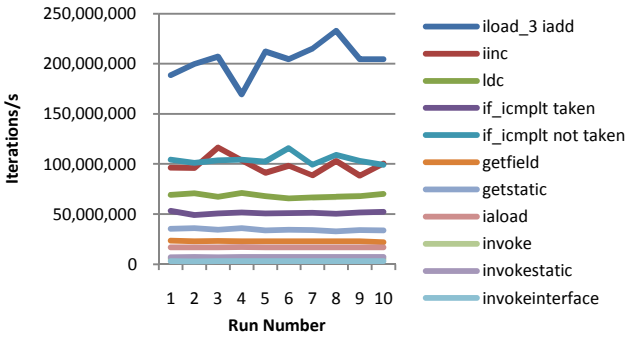

Figure 6: MIDP on Smart Phone

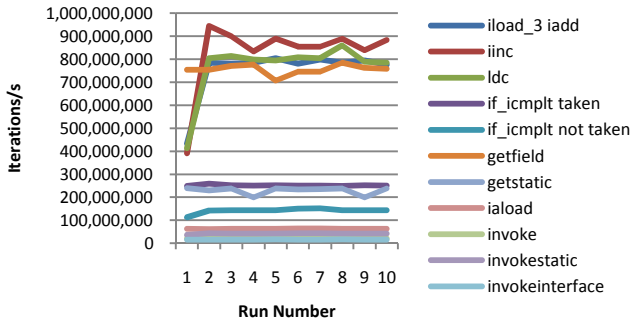

Figure 7: Fiji-VM on Laptop

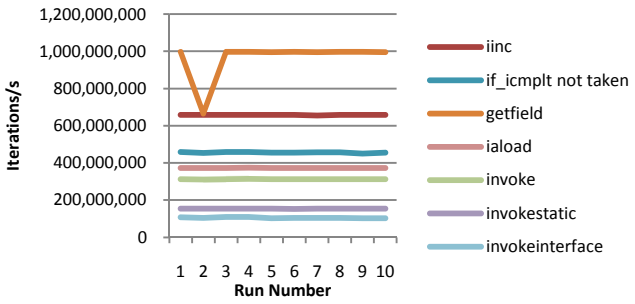

Figure 8: GCJ (Optimized) on Laptop

Figures 5-8 show the stability during multiple runs of the bytecode benchmarks to find that GCJ with optimizations during compilation is the most stable; followed by Fiji-VM, MIDP on the smart-phone, and finally the standard laptop jar executions. Despite targeting the CDC revision, these results are expected because we have worst case JIT through to AOT executions – which will be more deterministic. Fluctuations occur due to initial classloading as classes are kept in L1 or L2 cache rather than fetched from memory for future use. Garbage collection can also be changed in compiler options but was not investigated in this paper.

For the Sieve, Kfl, UDP/IP, and Lift benchmarks, we can directly compare the stability and results in Figures 9-11. Table 5 including a smart phone. 'X' denotes where information was unobtainable at the time of publication.

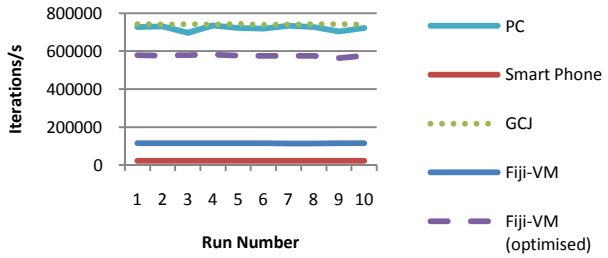

Figure 9: Sieve Benchmark Results

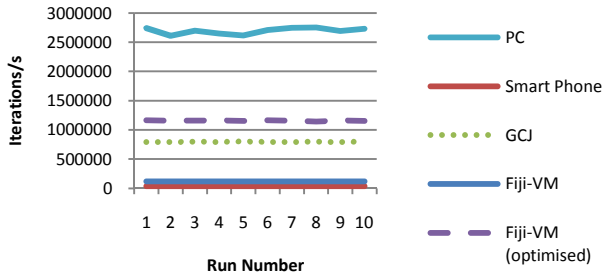

Figure 10: Kfl Benchmark Results

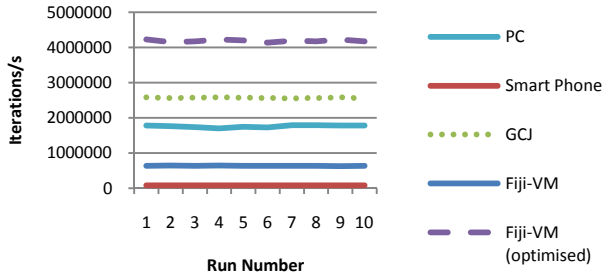

Figure 11: UDP/IP Benchmark Results

Table 5: RTJS Benchmark Applications

Platform	Speed (MHz)	Sieve	Kfl	UDP/IP	Lift
		(Thousands of Iterations/s)			
Standard JOP	100	7.386	16.591	6.527	1.255
LEON3 & JOP	40	1.394	4.810	2.595	4.721
aJile	165	X	X	X	X
PC & JRE 1.6	2x3000	771.011	2631	762.6	1974
PC & GCJ	2x2000	741.359	2566.9	794.569	X
PC & Fiji-VM	2x2000	575.513	4184.2	1159.3	X
Phone & MIDP	400	22.734	82.926	36.352	X
Phone & DVM	1000	X	X	X	X

The data for the standard JOP platform shown in Table 5 is taken from experimental results and published work in [31] and [32] for the standard JOP platform. The laptop runs a JRE at version 1.6 on 2 x Pentium 4 2GHz processors with 1 GB RAM and, as expected, is able to run large iterations of the micro benchmarks in comparison to the FPGA hardware implementations due to the high processor speed. These results are then normalized in Figure 12 to a standard 100 MHz JOP platform which is the original platform to compare against.

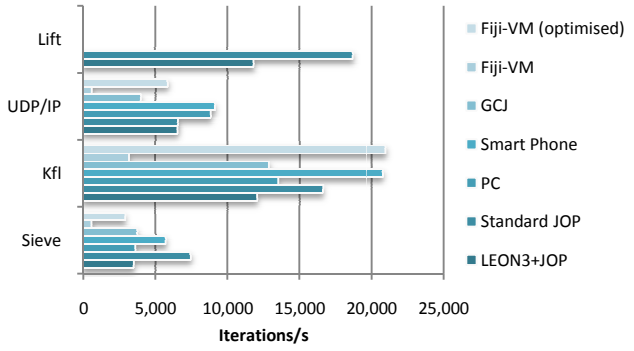

Figure 12: Normalized Comparison of Micro Benchmark Experiments

The normalized results shown in Figure 12 explore the overhead of running in the Java processor SoC design and compare them with other common Java platforms to find that the standard JOP platform outperforms the new LEON3 & JOP configuration by between 6% and 211%. The relative performance between the dual-processor and the standard JOP platform is -0.06% and -53%. This is highly dependent on how much memory access there is in a given application caused by slower I/O using direct memory access arbitration using the AHB bus and the parallel nature of the SoC design.

Both the new LEON3 & JOP configuration, as well as the standard JOP, outperform the ahead-of-time virtual machine implementations (to be confirmed) and the laptop but the fastest is the mobile phone running the CLDC Java stack. This is explained by the fact that the base software stack is smaller. As the target functionality for both FPGA implementations targets the CDC stack, a larger configuration, more time is spent in both class loading and

class searching in comparison to the smaller CLDC stack. However, in applications requiring greater Java functionality at the JVM level, the mobile phone with the smaller MID profile will be unsuitable while the LEON3 & JOP or JOP platform operates the larger foundation profile. The timing overhead is also dependant on the cache memory configurations used such as 1 KB in 4 blocks or as 16 KB in 64 blocks. This is investigated in [33] where larger cache sizes offer speeds of up to 28% but some applications seem to saturate at 4 KB cache and negligible performance improvement occurs with larger cache sizes.

Of the AOT compilations, the Fiji-VM implementation performs well in comparison to the hardware implementations. Given the ease of control through compiler arguments, the time spent to implement and execute the testbenches was also much lower in comparison to hardware JVMs such as the LEON3&JOP or JOP platforms. To guarantee real-time functionality, further tests on the classloader and garbage collectior are required.

5. LIMITATIONS & RESEARCH CHALLENGES

In this paper, Java technologies and compilation tools towards agent middleware and real-time applications have been investigated. But there are limitations and challenges to both hardware and software based implementations.

5.1. Java Execution on Hardware-based JVMs

Native Functions: During experimentation on jop-sim, many classes were missing from JOP's Java runtime environment to run the agent middleware application. These 683 class files were identified and were added to the agent middleware runtime at a cost of 219 kB, a 42% increase in footprint size, using the extensions classloader or classpath method of classloading. Once this was achieved, the JADE-FT application fails due to stack overflow errors. There are two ways to try and overcome this error: 1) increase the JOP stack and 2) investigate the call depth of the agent middleware application methods which are loaded onto the stack. JOP's stack size was increased to 8 KB and a further reduction of the missing runtime classes from 683 to 331 (89% more classes down to a 43% increase) did not prove successful.

Call Depth: An application to load agent services and start agent systems was tested on JADE and MicroFIPA-OS to investigate the combined application and runtime environment call depth; i.e. which methods were native JVM calls and which were JADE calls on an emulated CDC1.0 using a laptop. This is shown in Figure 13 for JADE 3.5 and MicroFIPA-OS respectively.

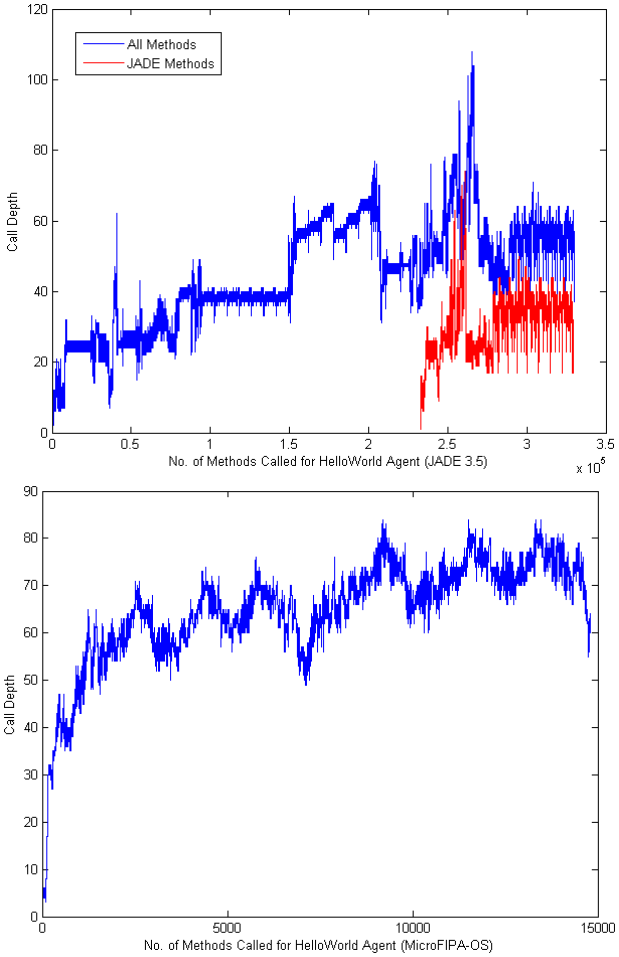

Figure 13: JADE and MicroFIPA-OS Call Depth Test Experiment

Figure 13 shows the cumulative number of calls and their origins. The JRE runtime prerequisite calls to run the test application account for 84.44% of all the calls made, with the final 15.66% being the middleware and application services.

Now, JOP's stack contains five words on return information, method arguments as first local variables, other public local variables, and the operand stack for expression evaluation per method invocation. The average stack frame of a native JOP method for these functions is typically around 10 to 20 bytes so with a call depth of 108, the resultant stack size need to be 2.16 kB. But existing JRE methods are much larger up to 256 bytes which gives a resultant stack size of 27.648 kB – far too large for the Spartan FPGA solution. Unless these existing method sizes are accounted for, the methods could not be pushed on to the stack and overflow. A larger FPGA board is required to confirm the final operation. The three key issues are: 1) the maximum call depth, 2) a reduced code size, and 3) embedded programming practices which all need further investigation. This is a significant finding and limitation of the combining current state-of-the-art agent technologies in software with hardware JVMs and on-chip memories towards real-time Agent computing.

This was not an issue however for the aJile-102 device with 32 kB D&I caches and its own executable optimizations, shown in Figure 14.

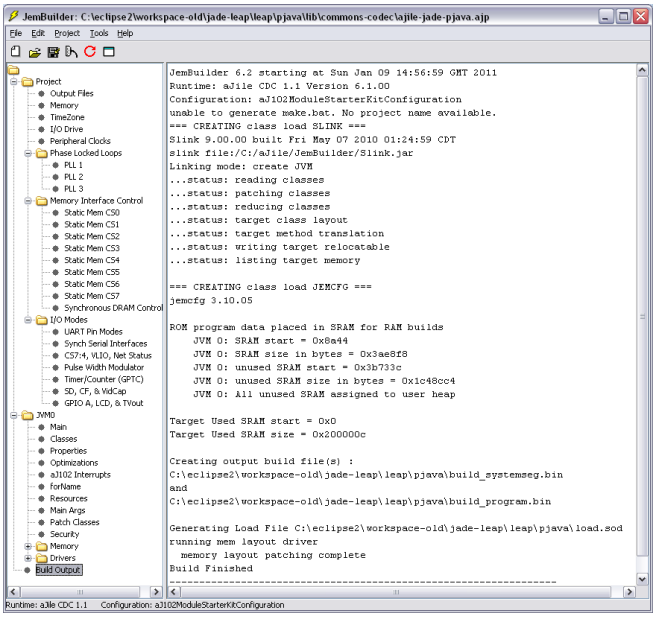

Figure 14: aJile-102 Compilation Screen of JADE-FT middleware application

Future steps in using JOP with the Agent middleware would be to re-implement some of the key functionalities into Native methods. This would use the standard bytecodes that JOP implements and thus reducing the classloading required.

5.2. Ahead of Time Software JVM Implementations

Startup Time: The JADE-FT implements an *instance manager* which was debugged to investigate which classes are first initialized. Key findings show that there is much class loading going on before the Agent middleware is initialized, shown in Figure 15. The exact locations of these middleware classes are delayed by hundreds of runtime environment classes caused by the representative Java runtime environment or virtual machine.

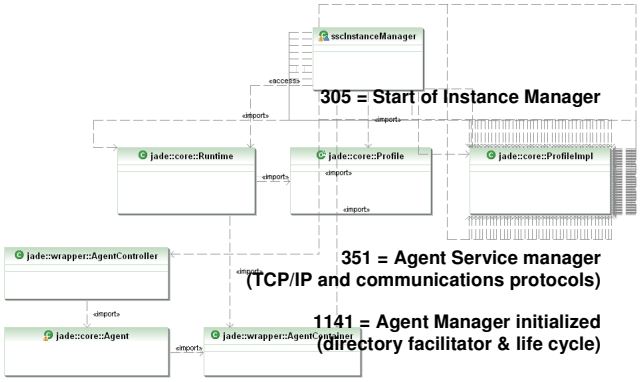

Figure 15: Instance Manager & Execution Times

A key limitation is the initial startup delay of the agent middleware due to the time taken to load the larger static class file application and associated libraries. Over 300 classes were loaded before the actual main() application is started and an examination of the boot-classloader could be performed to alleviate this issue.

Security: As dynamic class-loading and code mobility are utilized in more complex distributed computing tasks, security and trusted libraries become an important design metric (as found across the Internet). One method could be to separate the one classloader into 3 separate classloaders in software for managing and protecting code/data sets, each with different security levels, that loads:

1. Instance Manager + Middleware (low risk)
2. Services (medium risk)
3. Applications (high risk)

The standard *DynamicClassLoader* method could also be followed because to unload just 1 method else the whole classloader must be unloaded ().

6. CONCLUSIONS

This paper has investigated the potential uses of multiple Java-based technologies, from Java processors such as JOP in an FPGA and the aj-102 ASIC to ahead-of-time compilers such as GCJ and Fiji-VM, to find that there is a trade off in performance and flexibility. The performance metrics are divided into static and dynamic measures to cover the static footprint, agent functions, memory usage, and timing using real-time testbench applications and a full agent middleware solution with instance management for communication and data distribution services. The metrics tested showed that most solutions are under 2 MB of static memory and aim to provide a starting point for either hardware or software development, together with the limitations. Both hardware-based Java virtual machines and software-based runtime environments have shown that there are a number of computing solutions available that could speed up design and comply to varying real-time distributed satellite missions utilizing networking. In particular, the runtime environment is noted at 84.44% of the startup time for the full agent middleware application as methods are loaded into cache with room for minimization. Extracting this information from hardware for further tests is an ongoing research issue.

REFERENCES

[1] C. Jackson, C. Smithies, M. Sweeting, "NASA IP Demonstration in-orbit via UOSAT-12 Minisatellite", in Proc. for 52nd International Astronautical Congress (IAC '01), Toulouse, France

[2] C. Matarasso, D. Giggenbach, F. Hermanns, and E. Lutz, "Multimedia Satellite Communications Experiments To The International Space Station", Int'l J. of Satellite Communications, 2002, No. 20, pp. 333-345

[3] Space Daily, "AlSAT-1 DMC Working Well In Orbit With First Use Of IP", Website, www.spacedaily.com/news/internet-02p.html (last accessed: 08.07.2009)

[4] C. P. Bridges, "Agent Computing Platform for Distributed Satellite Systems", PhD Thesis, University of Surrey, 2010

[5] JADE – Java Agent Development Framework, Website, jade.tilab.com/ (last accessed: 17.06.2009)

[6] FIPA-OS at Sourceforge, Website, source-forge.net/projects/fipa-os/ (last accessed: 17.06.2009)

[7] T. Schetter, M. Campbell, and D. Surka, "Multiple Agent-Based Autonomy for Satellite Constellations", Artificial Intelligence Archive, Vol. 145, Issue 1-2, April 2003

[8] RTSJ Real-time Java Specification Main Webpage [Online], Available: www.rtsj.org/ (last accessed: 17.06.2009)

[9] Information Society Technologies, "HIJA – Safety Critical Java", Proposal (Revision 0.5), Website, www.aicas.com/papers/scj.pdf (last accessed 01.06.2010)

[10] F. Brandner, T. Thorn, and M. Schoeberl, "Embedded JIT compilation with CACAO on YARI," Institute of Computer Engineering, Vienna, Austria, June 2008

[11] Free Software Foundation, "The GNU Compiler for the JavaTM Programming Language" [Online], Available: http://gcc.gnu.org/java/ (last accessed 01.06.2010)

[12] M. Schoeberl, "A Java processor architecture for embedded real-time systems", J. of Systems Architecture, DOI: 54/1--2:265--286, 2008

[13] M. Zabel, T.B. Preußer, P.Reichel, and R.G. Spallek , "SHAP – Secure Hardware Agent Platform", in Proc. of the Dresdner Arbeitstagung Schaltungs und Systementwurf, 2007, pp.119-126

[14] T. Garrison, M. Ince, J. Pizzicaroli, and P. Swan, "IRIDIUM Constellation Dynamics, The Systems Engineering Trades" in the AIAA Proc. for the 46th International Astronautical Congress (IAC '95), Oslo, Norway

[15] B. Palmintier, R. Twiggs, and C. Kitts, "Distributed Com-puting on Emerald: A modular approach for Robust Distri-buted Space Systems", in Proc. IEEE Aerospace Conference (IEEEAC '00), pp. 211-222

[16] ESA Website: Cluster Mission, Website, sci.esa.int/cluster (last accessed: 08.07.2009)

[17] S. Chien, R. Sherwood, M. Burl, R. Knight, G. Rabideau, B. Engelhardt, and A. Davies, "The Techsat-21 Autonomous Sciencecraft Constellation", in Proc. of the Autonomous Agents and Multi-Agent Systems Conference, 2002

[18] M. Maleski and A. E. Zimbler, "Internetworking through Milstar" in Proc. for Military Communications Conference (MILCOM '95), pp. 474-478

[19] Candace C. Carlisle, and Eric J. Finnegan, "Space Technology 5: Pathfinder for Future Micro-Sat Constellations", in Proc. IEEE Aerospace Conference (IEEEAC '04), pp. 227-239

[20] Orbital Fact Sheet on FORMOSAT-3, Datasheet, www.orbital.com/NewsInfo/Publications/FORMOSAT_FS.pdf (last accessed 08.07.2009)

[21] S. Persson, S. D'Amico, J. Harr, "Flight Results from PRISMA Formation Flying and Rendevous Demonstration Mission", Proc. of 61st International Astronautical Congress, Prague, CZ, Paper ID: IAC-10-D9.2.8

[22] The von Karman Institute for Fluid Dynamics, "QB50, an international network of 50 CubeSats for multi-point, in-situ measurements in the lower thermosphere and re-entry research", Website: http://www.vki.ac.be/QB50/project.php

[23] Defence Agency Research Projects Agency, "DARPA Awards Contracts for Fractionates Spacecraft Program", News Release, 26th February 2008

[24] S. D'Amico, E. Gill, M. Garcia, O. Montenbruck, "GPS-Based Real-Time Navigation for the PRISMA Formation Flying Mission" in Proc. of NAVITEC'2006, 11-13 December 2006, Noordwijk (2006)

[25] T. Vladimirova, C. P. Bridges, J. R. Paul, S. A. Malik, and M. N. Sweeting, "Space-based Wireless Space Networks: Design Issues", *Proc. of IEEE Aerospace Conference 2010*, Big Sky, USA (IEEEAC'10)

[26] aJile Systems Inc, "aJ-102 Evaluation Kit", Found online at: www.ajile.com/downloads/aJ-102EK.pdf (Last Accessed: 21.05.2010)

[27] C.P. Bridges and T. Vladimirova, "Agent Computing Applications in Distributed Satellite Systems", in Proc. for the International Symposium for Autonomous Decentralised Systems (ISADS '09), Athens, Greece

[28] Google Inc., "Nexus One – Google Phone Gallery", Website, http://www.google.com/phone/detail/nexus-one (last accessed: 26.10.2010)

[29] T. Vladimirova, X. Wu, A.H. Jallad and C.P. Bridges, "Distributed Computing in Reconfigurable Picosatellite Networks", in Proc. of the 2nd NASA/ ESA Conference on Adaptive Hardware and Systems (AHS '07), pp. 682-692

[30] M. Schoberl, “JavaBenchEmbedded V1.0”, Website, www.jopdesign.com/perf.jsp (last accessed: 02.07.2009)

[31] B M. Schoeberl, “Time-predictable Computer Architecture”, EURASIP J. on Embedded Systems, Vol. 2009, Article ID 758480

[32] F. Brandner, T. Thorn, and M. Schoeberl, “Embedded JIT compilation with CACAO on YARI”, in Proc. of the 12th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2009), Tokyo, Japan

[33] M. Schoeberl, W. Binder, P. Moret, and A Villazon, “Design space exploration for Java processors with cross-profiling”, in Proc. of the 6th International Conference on the Quantitative Evaluation of SysTems (QEST 2009), Budapest, Hungary

ACKNOWLEDGEMENTS

This work has been supported by the EPSRC PhD+ Grant.

BIOGRAPHY

Christopher P. Bridges is a Research Fellow in the Astrodynamics Group at Surrey Space Centre. He gained his BEng in Electronic Engineering from the University of Greenwich and completed his PhD in ‘Agent Computing for Distributed Satellite Systems’ at the University of Surrey in 2009. He won the PhD+ scholarship grant to continue work on CubeSat computing payloads and Java technologies and has since started in the Astrodynamics Group on mission analysis and a visual inspection payload.

Tanya Vladimirova, MEng, MSc, PhD, CEng, MIET, MIEEE, is a Reader at the Department of Electronic Engineering, University of Surrey, UK. She is a member of the Surrey Space Centre, where she leads the VLSI Design and Embedded Systems research group. Her research interests are in the areas of FPGA/ASIC design, wireless sensor networks, image processing for embedded systems, distributed computing and artificial intelligence.