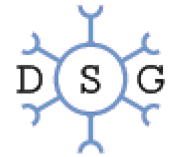
A Software Framework for Embedded Kernel Design and Development



Hong Ong and Mark Baker

Distributed Systems Group

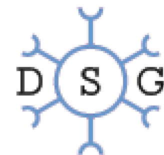
University of Portsmouth

hong.ong@port.ac.uk

http://dsg.port.ac.uk/hong

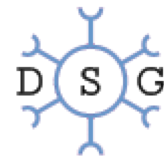
Invited talk, IBM, Hursley, UK. March 13th 2003

Outline

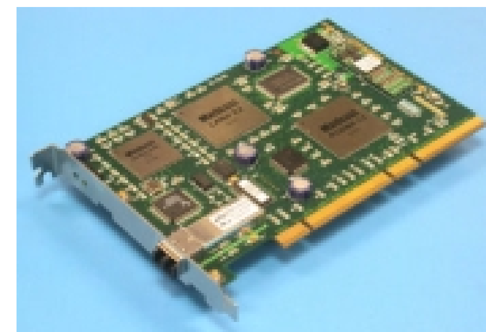


- Motivation.
- Design Principles.
- Implementation.
- Remaining Issues.
- Assessment and Future work.

History

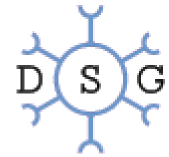


- Our original motivation was:
  - To design an Abstract Device Interface (ADI) for MPJ.
  - To improve the communications performance of MPJ.
- New opportunity:
  - Devices have more powerful processors and more memory.
    - For example, Myrinet - 8 Mbytes of RAM, 200Mhz RISC processor.
  - Can now offload protocol processing and also service processing.



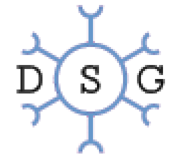
- New problem:
  - A runtime environment or Embedded Operating System (EOS) kernel is required to host and manage the offered services.

Distributions of Embedded OS



- MontaVista Software,
- LynuxWorks,
- Lineo,
- eCos (RedHat),
- PalmOS,
- Inferno,
- RTX,
- JavaOS,
  - Personal Java,
  - Embedded Java.
- Many others ...

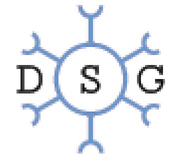
A Typical Embedded OS Architecture

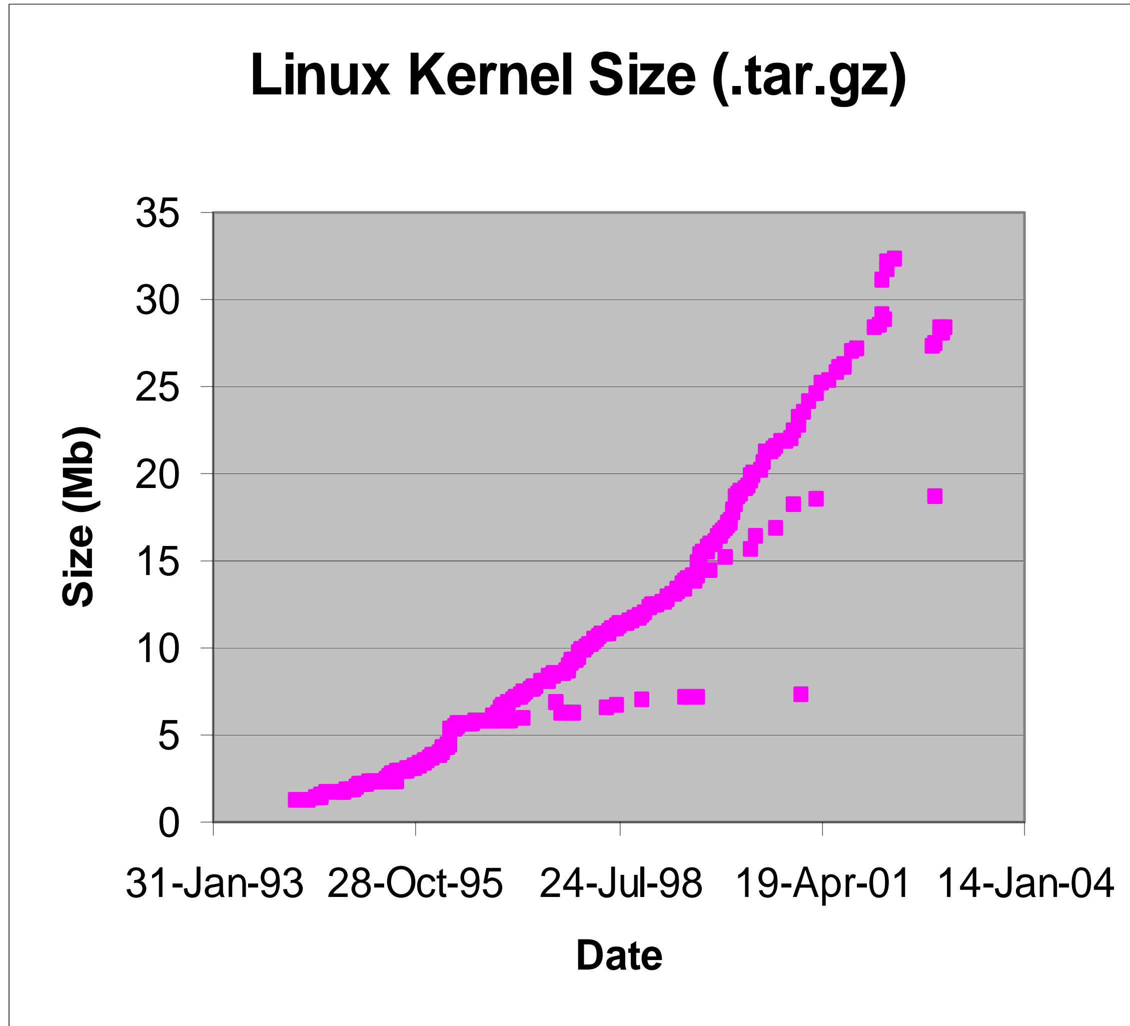


Java Application	Java Application	...	Java Application

Java APIs: Foundation classes, AWT, Network, ...	JNI API
Java Virtual Machine	Native Library
Embedded OS Kernel	
Booting, Traps, Interrupt, threads, ...	
Hardware	

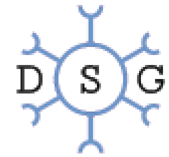
Why not Linux?





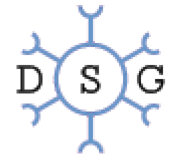
- Typical memory footprint is **1.5MB** (uncompressed)
- Can reduce to **260KB** with careful configuration (without networking support).
- Does not meet real-time requirements.

Why Java for Embedded OS?



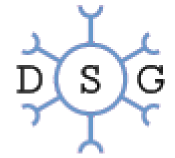
- Higher level of abstraction.
- Java is relatively easy to master, compared to C++, C, and ASM.
- Widespread use.
- Java supports dynamic loading of new classes.
- Designed to support code reuse.
- Support for distributed applications.
- Well-defined execution semantics.
- Java has built-in security infrastructure.
- Many other positive reasons ...

Shortcomings of Java for Embedded OS



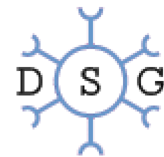
- Java has not been developed for embedded platforms:
  - No direct access to hardware.
    - No pointers or byte codes to access physical addresses.
- Inefficiencies:
  - Byte-code interpretation is slower than C/C++.
  - GC requires processor time.
- Unpredictability:
  - GC may pre-empt a running program.
  - Problems with real-time aspects.

JEM Project



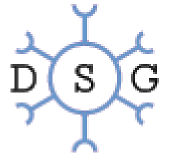
- Led us to the construction of JEM:
  - JEM = Java Embedded Micro-kernel design and development framework.
- Leverage designs and experiences from other research projects.
  - Kernel – L4, QNX, etc.
  - Safe language environment – J-Kernel, SPINE, etc.
  - UNET – Jaguar, PANIC, etc.
- A generic framework for developing kernels:
  - Embedded Operating System (EOS).
  - Embedded Applications (EA).

Design Principles



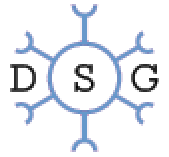
- Flexibility.
- Extensibility.
- Modularity.

Design Principles



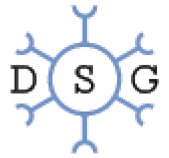
- Flexibility:
  - No predefined kernel philosophy.
    - Can build exo-kernel, micro-kernel, mono-kernel or application specific kernel.
  - No predefined kernel functionality.
    - Can build or reuse kernel services to accommodate different requirements.
    - Leverage the domain knowledge and effort of experienced developers.
      - Short development turn around time.

Design Principles



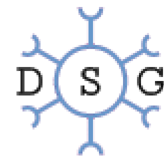
- Extensibility:
  - Assemble essential EOS services at compile time.
  - Other EOS and application services can be dynamically loaded at runtime.
  - Abstract Hardware Layer (AHL).
    - Separate language and hardware architecture.
    - Provide direct and safe access to hardware.

Design Principles



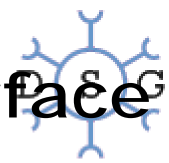
- Modularity:
  - Software module is the unit of deployment and configuration.
    - Can have variable granularity.
  - Encourages code-reuse.
  - Software quality can be improved by localizing the effects of design and implementation changes.
    - Data encapsulation.

Some Definitions

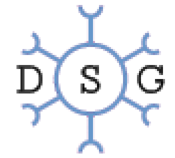


- A Module.
  - A finite set of well defined interfaces.
  - A finite set of global and private data.
  - A unit of configuration and deployment.
- An Interface.
  - A finite set of methods.
- A Binding.
  - The end result of establishing a communication channel between two or more objects.
- A Domain.
  - A finite set of modules.
  - A unit of protection.

JEM Framework



JEM Framework - Model



Trading Service Model

Client

3. Invoke an object (a binding)

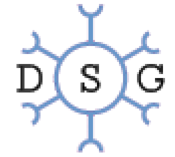
Server

2. Return query for service

1. Export the service

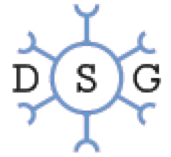
Broker

JEM Framework – Implementation



- Interfaces:
  - Reference,
  - Context,
  - Name,
  - Binding.

JEM Framework – Reference Interface

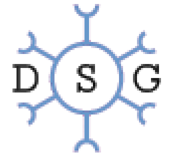


- Reference interface:

```
interface Reference {
}
```

  - The super class of all interfaces.
  - A “pointer” pointing to a named object containing all published methods.

JEM Framework – Name Interface

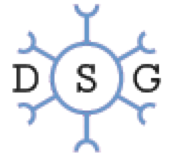


- Name interface:

```
interface Name {
    Context getContext();
    byte[] encode();
    …
}
```

  - The super class of all names.
  - An object is designated by a name.
  - Names are Context dependent.

JEM Framework – Context Interface

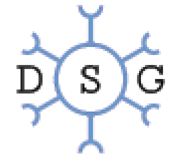


- A context is a set of mappings from names onto entities.

```
interface Context {
    Name export(Reference ref);
    Name decode(byte[] name);
    …
}
```

  - Contains:
    - A naming convention,
    - A name allocation policy.

JEM Framework – Binding Interface

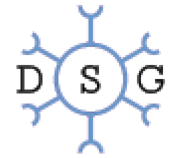


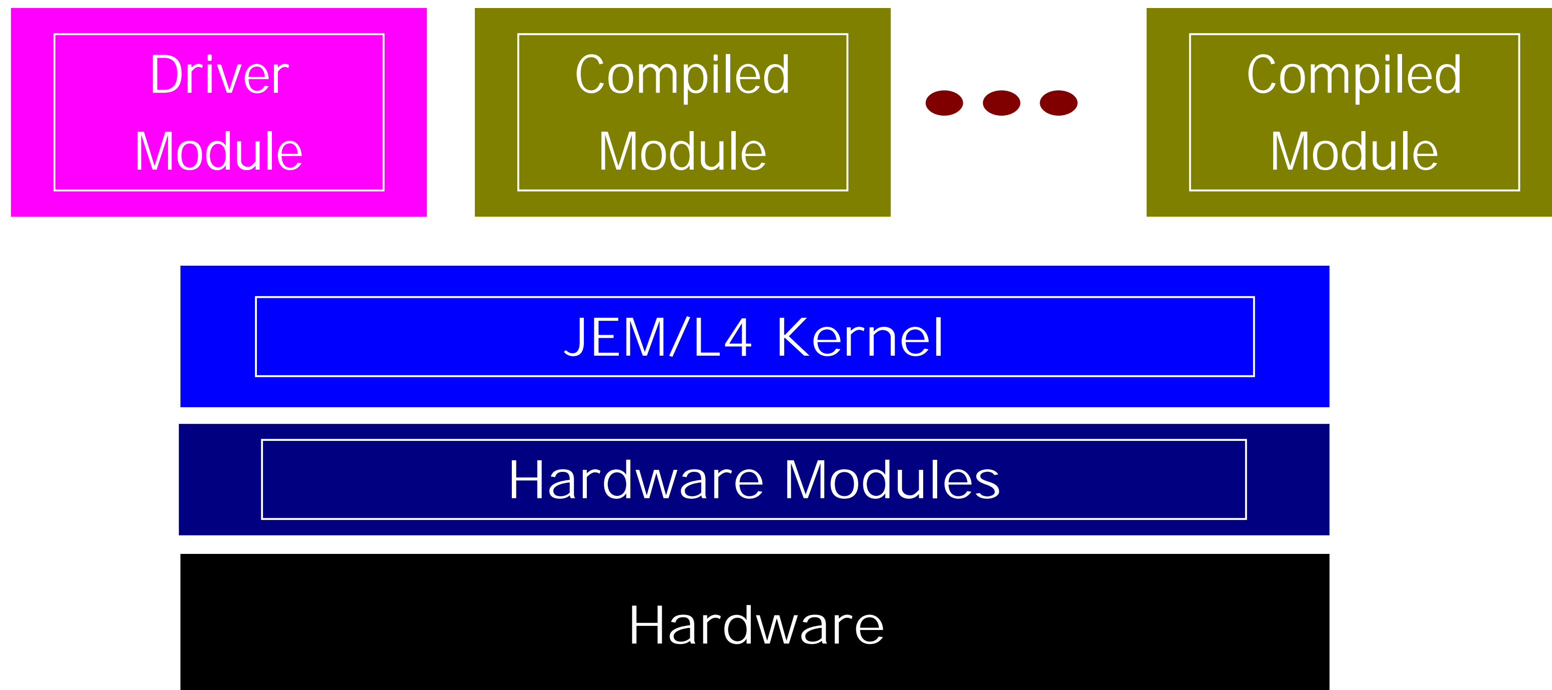
- A binding is a communication channel between modules.

```
interface Binding {
    Reference bind(Name name);
}
```

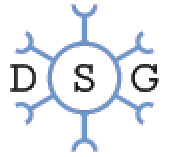
- Created either statically or dynamically through a Binding Manager.
- Supports arbitrary IPC and RPC semantics:
  - RMI, Message passing, etc ...
- Implemented as a module in JEM.

JEM/L4 Architecture





JEM/L4 Kernel



Driver Module

Service Processing Module

Service Processing Module

• • •

Binding

IPC Module

Thread Management Module

Memory Management Module

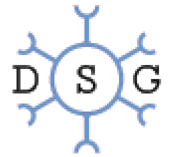
Binding

HW Clock Modules

• • •

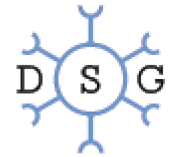
HW IRQ Module

JEM/L4 kernel – Implementation




```
<module name="L4">
<description>
   This module implemented the L4-like micro-kernel.
</description>
<file name="l4" />
<requires name="emu_boot" />
<requires name="emu_allocator" />
<requires name="emu_property" />
<requires name="emu_thread" />
<requires name="emu_trap" />
<requires name="roundrobin" />
<requires name="space" />
<requires name="allocator" />
<requires name="trader" />
<requires name="localbind" />
<requires name="sbrk" />
</module>
```

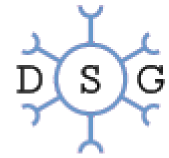
JEM/L4 Kernel - Implementation (cont')




```
/** L4-like Micro kernel */
#include <framework/jem.h>
unsigned char kernelstack[4096];
unsigned int kernelstacksize = 4096;

/** Main method call by JEM */
void prekernelstart(unsigned long offset) {}
void l4Probe(void) {  while(1); }
```

L4 Modules Dependency

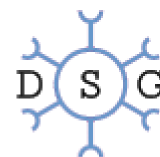


Kernel Graph Plotter : l4

Actions Modules

l4
roundrobin
emu_thread
allocator
sbrk
emu_allocator
space
emu_property
trader
emu_trap
emu_lowscheduler
localbind
emu_boot

Generating a Kernel



*.java
*.c
(JEM Modules)

Compilation Process

*.s
(assembly code)

as
(assembly compiler)

*.o
(object code)

*.a
(runtime library)

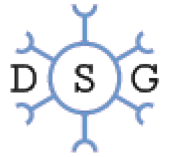
ld
(linker)

Native code
(executable)

Loader

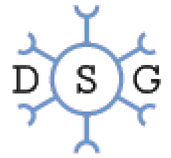
Myrinet 2000
Network
Interface

Tools for Generating a Kernel



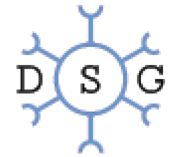
- Configuration tools:
  - Module dependency checking.
    - Determine the necessary modules by reading meta-data written in an XML dialect.
  - Module probing.
    - Extends the IBM Jikes compiler to generate C "stub" code to glue the modules together.
- Build tool:
  - Apache Ant 1.5.
    - Module dependency task and Module Probe task.

Tools for Generating a kernel (cont')



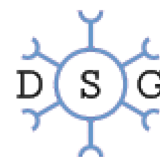
- Compilation tools:
  - J2C, GCJ. GCC.
- Linker tool:
  - Processor specific; GCC ld.
- Loader tool:
  - Processor specific; takes executable and places instructions and data into target;
- Libraries.
  - C library, Java runtime library, etc.
- Visualization tool:
  - A Java graphical tool for drawing module dependency graphs.

JEM Modules – Current Status



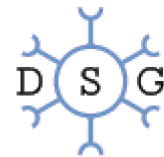
- Binding module:
  - Local (pointer to interface descriptor).
  - Syscall (client stub perform a trap).
- Language modules:
  - C library.
  - Java runtime library.
- Hardware modules:
  - Boot.
  - Exception.
- Kernel modules:
  - Memory Management:
    - Paged and Buddy.
  - Interrupt:
    - PIO, PIC.
  - Thread and Scheduler:
    - Thread, Round-Robin, Priority, Bitmap.
  - Network:
    - RPC, TCP/IP, UDP/IP.

Remaining Issues



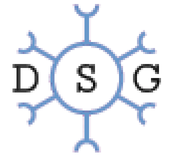
- The configuration process:
  - How to guide developer in choosing the right module?
  - How to analyze the resulting composition of modules?
    - Are they correct and meet the desired requirements?
- Software development:
  - How to develop lightweight interface?
  - How to define metrics and develop techniques for categorizing modules against memory size, security, and QoS?
  - How to develop and save configuration information?

Remaining Issues (cont')



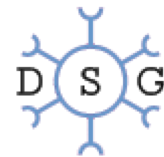
- Re-configurable hardware:
  - What is the impact on an OS and application services for both functional and non-functional attributes?
  - How to determine when the performance gain is worth the cost?

Assessment and Future work



- It is feasible to use the trading service model for developing EOS kernel.
- Java can be used as a programming tool for embedded systems.
  - But it is not without hurdles.
    - For example, implementing the native portions of the Java runtime library.
  - C and Assembly are still useful/necessary.
- Future work:
  - Benchmarking.
  - Developing more modules.
  - Addressing some of the remaining issues.

Information



- DSG URL
  - http://dsg.port.ac.uk
- JEM URL:
  - http://dsg.port.ac.uk/projects/research/JEM
- E-mails:
  - Hong.Ong@port.ac.uk
  - Mark.Baker@computer.org

QUESTIONS ?