technische universität
dortmund

fakultät für informatik
informatik 12

Imperative languages

Peter Marwedel
TU Dortmund,
Informatik 12

2008/10/28

Models of computation considered in this course

Communication/ local computations	**Shared memory**	**Message passing**	
		Synchronous	**| Asynchronous**
Communicating finite state machines	StateCharts		SDL
Data flow model ⊂ Computational graphs	Not useful	Simulink Sequence dia-gram, Petri nets	Kahn process networks, SDF
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	
Discrete event (DE) model	VHDL, …	Only experimental systems, e.g. distributed DE in Ptolemy	

Java (1)

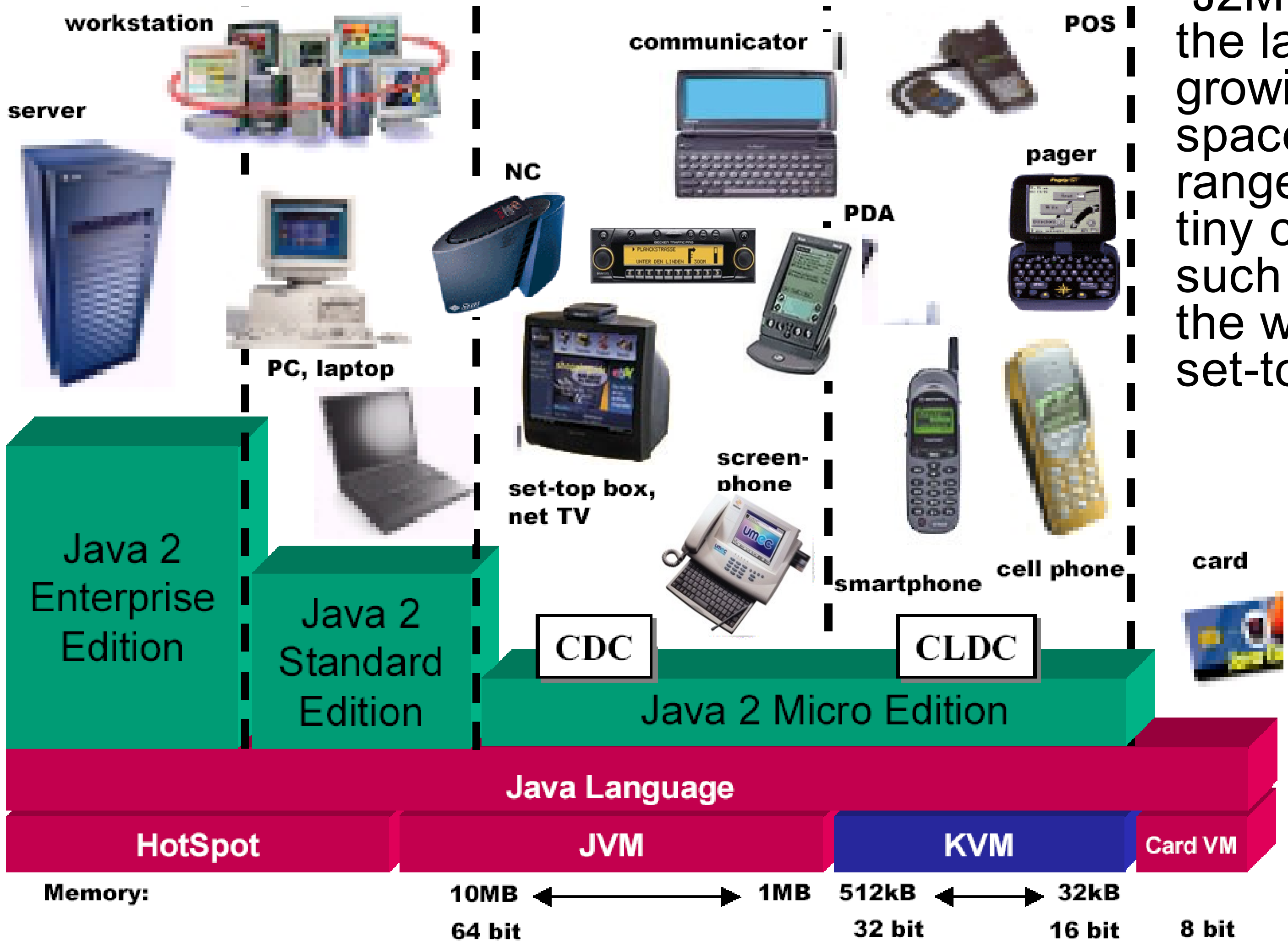
Potential benefits:

- Clean and safe language
- Supports multi-threading (no OS required?)
- Platform independence (relevant for telecommunications)

Problems:

- Size of Java run-time libraries? Memory requirements.
- Access to special hardware features
- Garbage collection time
- Non-deterministic dispatcher
- Performance problems
- Checking of real-time constraints

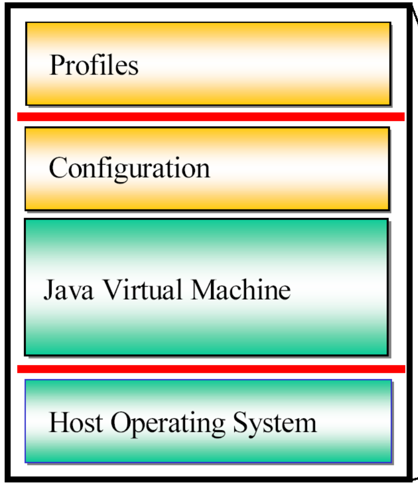
Overview over Java 2 Editions



"J2ME … addresses the large, rapidly growing consumer space, which covers a range of devices from tiny commodities, such as pagers, all the way up to the TV set-top box.."

Based on http://java.sun.com/products/cldc/wp/KVMwp.pdf

Software stack for J2ME



- ***Java Virtual Machine:*** implementation of a Java VM, customized for a particular device's host OS and supports a particular J2ME configuration.
- ***Configuration:*** defines the minimum set of Java VM features and Java class libraries available on a particular "category" of devices representing a particular "horizontal" market segment.
  In a way, a configuration defines the "lowest common denominator" of the Java platform features and libraries that the developers can assume to be available on all devices.
- ***Profile:*** defines the minimum set of Application Programming Interfaces (APIs) available on a particular "family" of devices representing a particular "vertical" market segment. Profiles are implemented "upon" a particular configuration. Applications are written "for" a particular profile and are thus portable to any device that "supports" that profile. A device can support multiple profiles.

Based upon
http://java.sun.com/products/cldc/wp/KVMwp.pd

KVM and CLDC

- ***The K Virtual Machine:***
  Highly portable Java VM designed for small memory, limited-resource, network-connected devices, e.g.: cell phones, pagers, & personal organizers. Devices typically contain 16- or 32-bit processors and a minimum total memory footprint of ~128 kilobytes.
- ***Connected, Limited Device Configuration (CLDC)***
  Designed for devices with intermittent network connections, slow processors and limited memory – devices such as mobile phones, two way pagers and PDAs. These devices typically have either 16- or 32-bit CPUs, and a minimum of 128 KB to 512 KB of memory.

CDC Configuration and MIDP 1.0 + 2.0 Profiles

- ***CDC:*** Designed for devices that have more memory, faster processors, and greater network bandwidth, such as TV set-top boxes, residential gateways, in-vehicle telematics systems, and high-end PDAs. Includes a full-featured Java VM, & a larger subset of the J2SE platform. Most CDC-targeted devices have 32- bit CPUs & ≥ 2MB of memory.
- ***Mobile Information Device Profile*** **(MIDP):**
  Designed for mobile phones & entry-level PDAs.
  Offers core application functionality for mobile applications, including UI, network connectivity, local data storage, & application management. With CLDC, MIDP provides Java runtime environment leveraging capabilities of handheld devices & minimizing memory and power consumption.

Real-time features of Java

J2ME, KVM, CLDC & MIDP not sufficient for real-time behavior. Real-time specification for Java (JSR-1) addresses 7 areas:

1. Thread Scheduling and Dispatching
2. Memory Management:
3. Synchronization and Resource Sharing
4. Asynchronous Event Handling
5. Asynchronous Transfer of Control
6. Asynchronous Thread Termination
7. Physical Memory Access

Designed to be used with any edition of Java.

[//www.rtj.org] [https://rtsj.dev.java.net/rtsj-V1.0.pdf]

Example: different types of memory areas

Area of memory may be used for the allocation of objects.

There are four basic types of memory areas (partially excluded from garbage collection)**:**

1. Scoped memory provides a mechanism for dealing with a class of objects that have a lifetime defined by syntactic scope.
2. Physical memory allows objects to be created within specific physical memory regions that have particular important characteristics, such as memory that has substantially faster access.
3. Immortal memory represents an area of memory containing objects that, once allocated, exist until the end of the application, i.e., the objects are immortal.
4. Heap memory represents an area of memory that is the heap. The RTSJ does not change the determinant of lifetime of objects on the heap. The lifetime is still determined by visibility.

[https://rtsj.dev.java.net/rtsj-V1.0.pdf]

Message passing libraries

Example: MPI/Open MPI

- Library designed for high-performance computing (hpc)
- Based on asynchronous/synchronous message passing
- Comprehensive, popular library
- Available on a variety of platforms
- Considered also for multiple processor system-on-a-chip (MPSoC) programming for embedded systems;
- MPI includes many copy operations to memory ☹ (memory speed ~ communication speed for MPSoCs); Appropriate MPSoC programming tools missing.

http://www.mhpcc.edu/training/workshop/mpi/MAIN.html#Getting_Started

MPI (1)

Sample blocking library call (for C):

- MPI_Send(*buffer,count,type,dest,tag,comm*) where
  - *buffer*: Address of data to be sent
  - *count*: number of data elements to be sent
  - *type*: data type of data to be sent (e.g. MPI_CHAR, MPI_SHORT, MPI_INT, …)
  - *dest*: process id of target process
  - *tag*: message id (for sorting incoming messages)
  - *comm*: communication context = set of processes for which destination field is valid
  - function result indicates success

http://www.mhpcc.edu/training/workshop/mpi/MAIN.html#Getting_Started

MPI (2)

Sample non-blocking library call (for C):

- MPI_Isend(*buffer,count,type,dest,tag,comm,request*) where
  - *buffer … comm:* same as above
  - *request*: the system issues a unique "request number". The programmer uses this system assigned "handle" later (in a WAIT type routine) to determine completion of the non-blocking operation.

http://www.mhpcc.edu/training/workshop/mpi/MAIN.html#Getting_Started

Evaluation

Explicit

- Computation partitioning
- Communication
- Data distribution

Implicit

- Synchronization (implied by communic., explicit possible)
- Expression of parallelism (implied)
- Communication mapping

Properties

- Most things are explicit
- Lots of work for the user ("*assembly lang. for parallel prog*.")
- doesn't scale well when # of processors is changed heavily

Based on Wilfried Verachtert (IMEC): *Introduction to Parallelism,* tutorial, DATE 2008

Ptheads

- Shared memory model
  - Completely explicit synchronization
  - Originally used for single processor
  - Exact semantics depends on the memory consistency model
  - Synchronisation is very hard to program correctly
- Consists of standard API
  - Locks (mutex, read-write locks)
  - Condition variables
  - Typically supported by a mixture of hardware (shared memory) and software (thread management)
- Support for efficient producer/consumer parallelism relies on murky parts of the model
- Pthreads can be used as back-end for other programming models (e.g. OpenMP)

Based on Wilfried Verachtert (IMEC): *Introduction to Parallelism,* tutorial, DATE 2008

PThreads Example

```
threads = (pthread_t *) malloc(n*sizeof(pthread_t));
pthread_attr_init(&pthread_custom_attr);

for (i=0;i<n; i++)
  pthread_create(&threads[i], &pthread_custom_attr, task,
…)

for (i=0;i<n; i++) {
  pthread_mutex_lock(&mutex);
  <receive message>
  pthread_mutex_unlock(&mutex);
}

for (i=0;i<n; i++)
  pthread_join(threads[i], NULL)
```

```
void* taks(void *arg) {
  …
  pthread_mutex_lock(&mutex);
  send message
  pthread_mutex_unlock(&mutex);
  return NULL
}
```

Based on Wilfried Verachtert (IMEC): *Introduction to Parallelism,* tutorial, DATE 2008

OpenMP

Explicit

- Expression of parallelism (mostly explicit)

Implicit

- Computation partitioning
- Communication
- Synchronization
- Data distribution

Parallelism expressed using pragmas

- Parallel loops (essentially data parallelism)
- Parallel sections
- Reductions

Implementations target shared memory hardware

Lack of control over partitioning can cause problems

Based on Wilfried Verachtert (IMEC): *Introduction to Parallelism,* tutorial, DATE 2008

Network Communication Protocols
- e.g. JXTA -

- *Open source peer-to-peer protocol specification.*
- *Defined as a set of XML messages that allow any device connected to a network to exchange messages and collaborate independently of the network topology.*
- *Designed to allow a range of devices to communicate. Can be implemented in any modern computer language.*
- *JXTA peers create a virtual overlay network, allowing a peer to interact with other peers even when some of the peers and resources are behind firewalls and NATs or use different network transports. Each resource is identified by a unique ID, so that a peer can change its localization address while keeping a constant identification number.*

http://en.wikipedia.org/wiki/JXTA

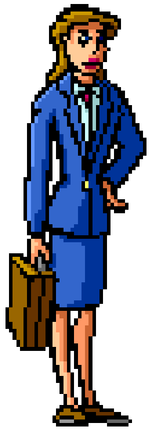
Network Communication Protocols - e.g. DPWS -

- *The **Devices Profile for Web Services** (DPWS) defines a minimal set of implementation constraints to enable secure Web Service messaging, discovery, description, and eventing on resource-constrained devices. …*
- *DPWS specifies a set of built-in services:*
  - *Discovery services: used by a device connected to a network to advertise itself and to discover other devices.*
  - *Metadata exchange services: provide dynamic access to a device's hosted services and to their metadata.*
  - *Publish/subscribe eventing services: allowing other devices to subscribe to asynchronous event messages*
- *Lightweight protocol, supporting dynamic discovery, … its application to automation environments is clear.*

http://en.wikipedia.org/wiki/Devices_Profile_for_Web_Services

Synchronous message passing: CSP

- **CSP** (communicating sequential processes) [Hoare, 1985], *rendez-vous*-based communication: Example:



```
process A
..
var a ...
  a:=3;
  c!a; -- output
end
```

```
process B
..
var b ...
  ...
  c?b; -- input
end
```

Synchronous message passing: ADA

After Ada Lovelace (said to be the 1st female programmer).

US Department of Defense (DoD) wanted to avoid multitude of programming languages

☞Definition of requirements

☞Selection of a language from a set of competing designs (selected design based on PASCAL)

ADA'95 is object-oriented extension of original ADA.

Salient: task concept

Synchronous message passing: Using of tasks in ADA

```
procedure example1 is
    task a;
    task b;
    task body a is
      -- local declarations for a
      begin
        -- statements for a
      end a;
    task body b is
      -- local declarations for b
      begin
        -- statements for b
      end b;
begin
  -- Tasks a and b will start before the first
  -- statement of the body of example1
end;
```

Synchronous message passing: ADA-rendez-vous

```
task screen_out is
 entry call_ch(val:character; x, y: integer);
 entry call_int(z, x, y: integer);
end screen_out;
task body  screen_out is
...
 select
  accept call_ch ...  do ..
  end call_ch;
 or
  accept call_int ... do ..
  end call_int;
 end select;
```

Sending a message:

```
begin
 screen_out.call_ch('Z',10,20);
 exception
  when tasking_error =>
              (exception handling)
end;
```

Other imperative languages

- **Pearl:** Designed in Germany for process control applications. Dating back to the 70s. Used to be popular in Europe.
  Pearl News still exists
  (in German, see http://www.real-time.de/)
- **Chill**: Designed for telephone exchange stations. Based on PASCAL.

Summary

Imperative languages

- Java
- MPI
- Pthreads
- OpenMP
- Network Communication Protocols
- CSP
- ADA
- Other languages