

Imperative languages

Peter Marwedel
TU Dortmund,
Informatik 12

2009/11/04



Models of computation considered in this course

Communication/ local computations	Shared memory	Message passing	
		Synchronous	Asynchronous
Undefined components		Plain text, use cases (Message) sequence charts	
Communicating finite state machines	StateCharts		SDL
Data flow	(Not useful)		Kahn networks, SDF
Petri nets		C/E nets, P/T nets, ...	
Discrete event (DE) model	VHDL, Verilog, SystemC, ...	Only experimental systems, e.g. distributed DE in Ptolemy	
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA	

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;
```

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

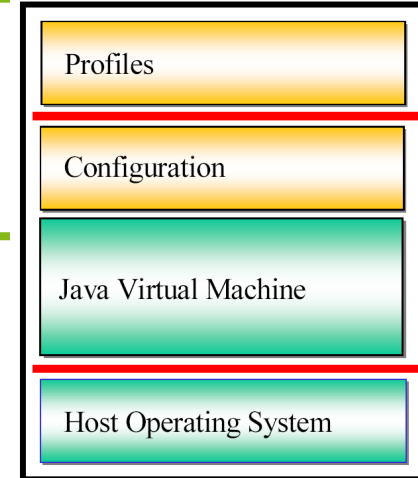
Overview over Java 2 Editions



“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 & ≥ 2 MB 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.

[<http://www.rtsj.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.
1. **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.
2. 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>]

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.

Communication libraries

- Communication libraries can add blocking or non-blocking communication to von-Neumann languages
- Examples will be presented in chapter 4

Summary

Imperative languages

- CSP
- ADA
- Java
- Other languages