Dataflow Languages

Purpose: concurrent or stream programming

Examples: SISAL, Strand, spreadsheets

Notable features: data driven computation

Dataflow Languages

Normally you think of a program as a sequence of operations to be done on some data

Dataflow Languages

Normally you think of a program as a sequence of operations to be done on some data

A dataflow language takes the view that the data should be the things of interest, and so the data drives the computations

Dataflow Languages

Normally you think of a program as a sequence of operations to be done on some data

A dataflow language takes the view that the data should be the things of interest, and so the data drives the computations

For example in `x = y + z` the addition can only be done when `y` and `z` have values

Dataflow Languages

So within

```
y = 1;
x = y + z;
z = 2;
```

the addition can only be executed *after* the assignments to both `y` and `z`, regardless of the order these statements happen to be written

Dataflow Languages

This may sound weird, but this is precisely how spreadsheets work

Dataflow Languages

This may sound weird, but this is precisely how spreadsheets work

Values propagate across cells by executing those rules that apply at any given point in time

Dataflow Languages

This may sound weird, but this is precisely how spreadsheets work

Values propagate across cells by executing those rules that apply at any given point in time

Regardless of the actual layout of the cells

Dataflow Languages

Outside spreadsheets there has not been a wide uptake of this approach, though it should be noted that event-driven programming is closely related

Dataflow Languages

Outside spreadsheets there has not been a wide uptake of this approach, though it should be noted that event-driven programming is closely related

There has been some experimentation in the area of parallel programming: note that applicable computations can be executed in parallel

Dataflow Languages

Outside spreadsheets there has not been a wide uptake of this approach, though it should be noted that event-driven programming is closely related

There has been some experimentation in the area of parallel programming: note that applicable computations can be executed in parallel

Exercise Deep Learning in AI has been called a dataflow approach. Read about this

Dataflow Languages

Feet

- Excel: You don’t need to shoot yourself in the foot because a macro virus has already done so

Markup Languages

Purpose: description of objects; often, but not exclusively, documents

Examples: HTML, XML, SGML, CSS, nroff, $\LaTeX$, . . .

Notable features: use of notation, e.g., within a document, to describe elements of the document (often, but not exclusively, visual layout); generally not “executed” in the usual sense

Markup Languages

- HTML: HyperText Markup Language
- XML: Extensible Markup Language
- SGML: Standard Generalized Markup Language
- CSS: Cascading Style Sheets
- nroff: new roff (roff: runoff)
- LaTeX: Lamport's TeX (TeX: from "technology")

Markup Languages

Feet

- HTML: You cut a bullethole in your foot with nothing more than a small penknife, but you realize that to make it look convincing, you need to be using Dreamweaver

Markup Languages

Feet

- HTML: You cut a bullethole in your foot with nothing more than a small penknife, but you realize that to make it look convincing, you need to be using Dreamweaver
- XML: You can’t actually shoot yourself in the foot; all you can do is describe the gun in painful detail

Markup Languages

Feet

- HTML: You cut a bullethole in your foot with nothing more than a small penknife, but you realize that to make it look convincing, you need to be using Dreamweaver
- XML: You can’t actually shoot yourself in the foot; all you can do is describe the gun in painful detail
- nroff:

  ```
  troff -ms -Hdrwp | lpr -Pwp2 & .*place
  bullet in footer .B .NR FT +3i .in 4 .bu Shoot!
  .br .sp .in -4 .br .bp NR HD -2i .*
  ```

Markup Languages

Feet

- HTML: You cut a bullethole in your foot with nothing more than a small penknife, but you realize that to make it look convincing, you need to be using Dreamweaver
- XML: You can't actually shoot yourself in the foot; all you can do is describe the gun in painful detail
- nroff:

  ```
  troff -ms -Hdrwp | lpr -Pwp2 & .*place
  bullet in footer .B .NR FT +3i .in 4 .bu Shoot!
  .br .sp .in -4 .br .bp NR HD -2i .*
  ```

- CSS: Everyone can now shoot themselves in the foot, but all their feet come out looking identical and attached to their ears

Markup Languages

Very widely used

- HTML was originally used to describe the content (“this is a section title”) and appearance (“use a big and bold font”) of Web pages. Usually poorly

Markup Languages

Very widely used

- HTML was originally used to describe the content (“this is a section title”) and appearance (“use a big and bold font”) of Web pages. Usually poorly
- These days the accepted approach is to use HTML to describe the content, and use CSS to describe the appearance

Markup Languages

Very widely used

- HTML was originally used to describe the content (“this is a section title”) and appearance (“use a big and bold font”) of Web pages. Usually poorly
- These days the accepted approach is to use HTML to describe the content, and use CSS to describe the appearance
- XML is used to markup the *meaning* of (say) text. Currently seen as the cure to all “Web 2.0” scenarios. Usually incorrectly

Markup Languages

HTML

```
<html>
<head>
<title>CM20318</title>
<link rel="stylesheet" type="text/css" href="notes.css">
</head>
<body>
<h2>CM20318: Comparative Programming Languages</h2>

<h4>Unit Catalogue</h4>

<a href="http://www.bath.ac.uk/catalogues/2023-2024/cm/CM20318.h
<p>
```

Markup Languages

CSS

```
body {
  font-family: Arial;
  background: white url("bg.png") repeat-y;
}

tt {
  font-size: larger;
}

.warn {
  color: red;
}
```

Markup Languages

XML

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
     "http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
      <m:OrderItemResponse xmlns:m="Some-URI">
      <OrderNumber>561381</OrderNumber>
      </m:OrderItemResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP is a standard data encoding for transfer of data between Web services that uses XML

Markup Languages

HTML

HTML and XML are both derivatives of a more general language, SGML

Markup Languages

HTML

HTML and XML are both derivatives of a more general language, SGML

HTML is ubiquitous

Markup Languages

HTML

HTML and XML are both derivatives of a more general language, SGML

HTML is ubiquitous

Unfortunately, its design ignored a lot of useful earlier work on hypertexts

Markup Languages

HTML

HTML and XML are both derivatives of a more general language, SGML

HTML is ubiquitous

Unfortunately, its design ignored a lot of useful earlier work on hypertexts

HTML/CSS is about *display* of documents

Markup Languages

HTML

HTML and XML are both derivatives of a more general language, SGML

HTML is ubiquitous

Unfortunately, its design ignored a lot of useful earlier work on hypertexts

HTML/CSS is about *display* of documents

Where we should mean ”display” in the general sense; including “audio display” for the vision-impaired

Markup Languages

XML

XML is about identifying information within documents

Markup Languages

XML

XML is about identifying information within documents

XML is widely used in Web 2.0 and Web services as data communication protocols

Markup Languages

XML

XML is about identifying information within documents

XML is widely used in Web 2.0 and Web services as data communication protocols

Specifications exists for:

Markup Languages

XML

XML is about identifying information within documents

XML is widely used in Web 2.0 and Web services as data communication protocols

Specifications exists for:

- MathML: mathematics

Markup Languages

XML

XML is about identifying information within documents

XML is widely used in Web 2.0 and Web services as data communication protocols

Specifications exists for:

- MathML: mathematics
- OFX: Open Financial Exchange, financial data

Markup Languages

XML

XML is about identifying information within documents

XML is widely used in Web 2.0 and Web services as data communication protocols

Specifications exists for:

- MathML: mathematics
- OFX: Open Financial Exchange, financial data
- XUL: XML User-interface Language, a language for describing user interfaces

Markup Languages

XML

XML is about identifying information within documents

XML is widely used in Web 2.0 and Web services as data communication protocols

Specifications exists for:

- MathML: mathematics
- OFX: Open Financial Exchange, financial data
- XUL: XML User-interface Language, a language for describing user interfaces
- AML: Astronomical Markup Language, for controlling astronomical instruments.

Markup Languages

XML

- RSS: Really Simple Syndication
- WML: Wireless Markup Language
- SVG: Scalable Vector Graphics
- MusicXML: music notation
- VoiceXML: Voice Extensible Markup Language
- PDML: Product Data Markup Language
- ODF: Open Document Format
- SMIL: Synchronized Multimedia Integration Language
- Gastro Intestinal Markup Language
- And hundreds of others

Markup Languages

Originally designed to be text based and therefore easily debugged by sight, common usage of HTML and XML is so complicated this is no longer possible

Markup Languages

Originally designed to be text based and therefore easily debugged by sight, common usage of HTML and XML is so complicated this is no longer possible

Being text based is now a disadvantage as it is hard for computers to parse quickly and accurately

Markup Languages

Originally designed to be text based and therefore easily debugged by sight, common usage of HTML and XML is so complicated this is no longer possible

Being text based is now a disadvantage as it is hard for computers to parse quickly and accurately

Not many humans read or write HTML these days

Markup Languages

Originally designed to be text based and therefore easily debugged by sight, common usage of HTML and XML is so complicated this is no longer possible

Being text based is now a disadvantage as it is hard for computers to parse quickly and accurately

Not many humans read or write HTML these days

XML has been adopted widely for Web applications, often without proper consideration of the alternatives, such as JSON (JavaScript Object Notation) or Google's Protocol Buffers

Markup Languages

Originally designed to be text based and therefore easily debugged by sight, common usage of HTML and XML is so complicated this is no longer possible

Being text based is now a disadvantage as it is hard for computers to parse quickly and accurately

Not many humans read or write HTML these days

XML has been adopted widely for Web applications, often without proper consideration of the alternatives, such as JSON (JavaScript Object Notation) or Google's Protocol Buffers

Exercise These are examples of the many *serialisation* languages: read about these

Markup Languages

Also, increasingly XML is being used to *store* information, which it is *very* ill suited to do

Markup Languages

Also, increasingly XML is being used to *store* information, which it is *very* ill suited to do

Use a database to store information!

Markup Languages

Also, increasingly XML is being used to *store* information, which it is *very* ill suited to do

Use a database to store information!

If you ever have a project that uses an “XML database”, walk away in disgust

Object Oriented Languages

Purpose: general programming

Examples: Java, C++, Objective C, C#, JavaScript, Eiffel, Swift . . . , and many other languages with objects of some kind

Notable features: use of objects as a means to control complexity

Object Oriented Languages

Purpose: general programming

Examples: Java, C++, Objective C, C#, JavaScript, Eiffel, Swift . . . , and many other languages with objects of some kind

Notable features: use of objects as a means to control complexity

The concept of objects is so persuasive that there are a large number of languages (Python, Haskell, etc.) that are not usually thought of as OO languages but incorporate objects in some way

Object Oriented Languages

Feet

- C++: You accidentally create a dozen clones of yourself and shoot them all in the foot. Emergency medical assistance is impossible since you can’t tell which are bitwise copies and which are just pointing at others and saying, “That’s me, over there.”

Object Oriented Languages

Feet

- C++: You accidentally create a dozen clones of yourself and shoot them all in the foot. Emergency medical assistance is impossible since you can’t tell which are bitwise copies and which are just pointing at others and saying, “That’s me, over there.”
- C++ (2): “C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows away your whole leg” (Bjarne Stroustrup)

Object Oriented Languages

Feet

- C++: You accidentally create a dozen clones of yourself and shoot them all in the foot. Emergency medical assistance is impossible since you can't tell which are bitwise copies and which are just pointing at others and saying, "That's me, over there."
- C++ (2): "C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows away your whole leg" (Bjarne Stroustrup)

Note that C++ is now such a large language ("many featured") that group projects using it often start by deciding on which subset of the language they are going to use

Object Oriented Languages

Feet

- Objective C: You write a protocol for shooting yourself in the foot so that all people can get shot in their feet

Object Oriented Languages

Feet

- Objective C: You write a protocol for shooting yourself in the foot so that all people can get shot in their feet
- C#: You can't figure out a different way to shoot yourself in the foot so you end up copying Java

Object Oriented Languages

Feet

- Objective C: You write a protocol for shooting yourself in the foot so that all people can get shot in their feet
- C#: You can’t figure out a different way to shoot yourself in the foot so you end up copying Java
- Eiffel: You take out a contract on your foot. The precondition is that there’s a bullet in the gun; the postcondition is that there’s a hole in your foot

Object Oriented Languages

Feet

- Swift: You try to shoot yourself in the foot with the ultra-modern Swift gun, but you discover the gun has no trigger. Instead, it's designed to shoot automatically only when pointed safely at its intended target, with any type of bullet. Occasionally it explodes in your hand and takes off your arm

Object Oriented Languages

Feet

- Swift: You try to shoot yourself in the foot with the ultra-modern Swift gun, but you discover the gun has no trigger. Instead, it's designed to shoot automatically only when pointed safely at its intended target, with any type of bullet. Occasionally it explodes in your hand and takes off your arm

  Swift is Objective-C without the C

Craig Federighi (Apple)

Object Oriented Languages

We are going to look at OO in depth later

Object Oriented Languages

We are going to look at OO in depth later

Warning: when you are talking to people and they use the word “object”, take care to make it clear if they are using the word in the OO sense, or in the generic (meaning just some “thing”) sense

Actor Languages

Related to OO is the concept of an *Actor*

Actor Languages

Related to OO is the concept of an *Actor*

An actor is an entity or object (sometimes as in the OO sense, sometimes not) that communicates with other actors purely by means of messages

Actor Languages

When an actor receives a message it may

Actor Languages

When an actor receives a message it may

- do some computation and modify its internal state

Actor Languages

When an actor receives a message it may

- do some computation and modify its internal state
- send some messages to other actors

Actor Languages

When an actor receives a message it may

- do some computation and modify its internal state
- send some messages to other actors
- create some new actors

Actor Languages

When an actor receives a message it may

- do some computation and modify its internal state
- send some messages to other actors
- create some new actors

And it might do any or all of these things in any order or at the same time in parallel

Actor Languages

When an actor receives a message it may

- do some computation and modify its internal state
- send some messages to other actors
- create some new actors

And it might do any or all of these things in any order or at the same time in parallel

A kind of dataflow on objects

Actor Languages

Purpose: general programming, simulation, concurrent systems

Examples: Pony, Erlang, Elixir (compiles to Erlang bytecode), Scala, but often added into a language by means of a library

Notable features: use of actors passing messages to structure systems

Actor Languages

The idea of actors was to decouple computation from communication, allowing a natural way of having asynchronous communication between computations

Actor Languages

The idea of actors was to decouple computation from communication, allowing a natural way of having asynchronous communication between computations

And to have an inherently concurrent way of programming that allows a huge amount of parallelism in the execution

Actor Languages

The idea of actors was to decouple computation from communication, allowing a natural way of having asynchronous communication between computations

And to have an inherently concurrent way of programming that allows a huge amount of parallelism in the execution

A system could have thousand or millions of actors and the language runtime will schedule them as it sees fit on the available hardware

Actor Languages

The idea of actors was to decouple computation from communication, allowing a natural way of having asynchronous communication between computations

And to have an inherently concurrent way of programming that allows a huge amount of parallelism in the execution

A system could have thousand or millions of actors and the language runtime will schedule them as it sees fit on the available hardware

But this means that such runtimes are incredibly complicated to make efficient

Actor Languages

The idea of actors was to decouple computation from communication, allowing a natural way of having asynchronous communication between computations

And to have an inherently concurrent way of programming that allows a huge amount of parallelism in the execution

A system could have thousand or millions of actors and the language runtime will schedule them as it sees fit on the available hardware

But this means that such runtimes are incredibly complicated to make efficient

Exercise A popular use of actors is in *Multi-Agent Systems*. Read about these

Language Families

And so on

Language Families

And so on

We could go on with more families (symbolic languages, probabilistic languages, etc.), but instead we shall change tack slightly to look more at the *features* that languages may support

Language Families

And so on

We could go on with more families (symbolic languages, probabilistic languages, etc.), but instead we shall change tack slightly to look more at the *features* that languages may support

So: we have looked at *what* languages might to; we turn to *how* they might do it

Other Classifications

There are many other classifications that cut across the families we have described

Other Classifications

There are many other classifications that cut across the families we have described

Some more important than others

Other Classifications

There are many other classifications that cut across the families we have described

Some more important than others

- Declarative and Imperative
- Parallel or Sequential
- GC and non-GC
- Strongly typed, weakly typed, statically typed, dynamically typed and untyped
- Area of application: numeric, symbolic, business process, graphical, database, . . .
- Interpreted and Compiled (byte code interpreted etc.)
- and so on

Declarative and Imperative

Imperative: the program describes the actions to be taken

Examples: C, Java, Lisp, Fortran, ...

Notable features: program code is essentially “do this; then this; then this”, with loops and functions, maybe sequential, maybe parallel and with all the things you are used to to control the flow of execution

Declarative and Imperative

Declarative: the program is a description of what we want, with little or no explicit direction on how to do it, or no particular control flow

Examples: Prolog, ASP (Answer Set Programming), Haskell, Mathematica (pattern matching part), SQL (Structured Query Language), configuration languages …

Notable features: the system itself determines how to progress a computation

Declarative and Imperative

Declarative: the program is a description of what we want, with little or no explicit direction on how to do it, or no particular control flow

Examples: Prolog, ASP (Answer Set Programming), Haskell, Mathematica (pattern matching part), SQL (Structured Query Language), configuration languages . . .

Notable features: the system itself determines how to progress a computation

For example, an SQL engine must find the best way of finding the records that fit the query

Declarative and Imperative

Terminology alert: some people say declarative languages are those languages where programs can be regarded as theorems with computations as the proofs of the theorems

Declarative and Imperative

Terminology alert: some people say declarative languages are those languages where programs can be regarded as theorems with computations as the proofs of the theorems

This would include functional programming and probably all other languages so is not such a helpful view for separating languages into different classifications

Declarative and Imperative

Feet

- Mathematica: You try to shoot yourself in the foot and then have to figure out why it didn’t work

Declarative and Imperative

Feet

- Mathematica: You try to shoot yourself in the foot and then have to figure out why it didn’t work
- Mathematica (2): Your code to shoot yourself in the foot actually shoots someone else in the foot, but you think it works because you still feel pain

Declarative and Imperative

Feet

- Mathematica: You try to shoot yourself in the foot and then have to figure out why it didn't work
- Mathematica (2): Your code to shoot yourself in the foot actually shoots someone else in the foot, but you think it works because you still feel pain
- SQL: You cut your foot off, send it out to a service bureau and when it returns, it has a hole in it but will no longer fit the attachment at the end of your leg

Declarative and Imperative

Imperative languages are clearly very widely used

Declarative and Imperative

Imperative languages are clearly very widely used

Declarative languages are also very widely used

Declarative and Imperative

Imperative languages are clearly very widely used

Declarative languages are also very widely used

This is because SQL is hugely widely used (it’s in your browser; it’s in your phone!)

Declarative and Imperative

One important subset of declarative languages are the logic languages such as Prolog and ASP

Declarative and Imperative

One important subset of declarative languages are the logic languages such as Prolog and ASP

Here the computations are definitely proofs of theorems

Declarative and Imperative

One important subset of declarative languages are the logic languages such as Prolog and ASP

Here the computations are definitely proofs of theorems

At a stretch, markup languages can be though of as declarative

Declarative and Imperative

One important subset of declarative languages are the logic languages such as Prolog and ASP

Here the computations are definitely proofs of theorems

At a stretch, markup languages can be though of as declarative

Programmers often have difficulty thinking in a declarative way, unless the problem is already declarative

Declarative and Imperative

One important subset of declarative languages are the logic languages such as Prolog and ASP

Here the computations are definitely proofs of theorems

At a stretch, markup languages can be though of as declarative

Programmers often have difficulty thinking in a declarative way, unless the problem is already declarative

But declarative languages are naturally parallel as they don’t describe sequences of operations

Declarative and Imperative

In a declarative language the system determines how to progress a computation

Declarative and Imperative

In a declarative language the system determines how to progress a computation

An SQL example. A database contains a table `Exams`

Name	Course	Mark
Smith	C++ Prog	65
Jones	C++ Prog	85
Brown	Java Prog	35
Smith	Java Prog	88

Declarative and Imperative

In a declarative language the system determines how to progress a computation

An SQL example. A database contains a table `Exams`

Name	Course	Mark
Smith	C++ Prog	65
Jones	C++ Prog	85
Brown	Java Prog	35
Smith	Java Prog	88

SQL query:

```
select Name, Mark from Exams where Course = 'C++ Prog' and
Mark > 50;
```

Declarative and Imperative

It returns something like

Name	Mark
Jones	85
Smith	65

Declarative and Imperative

It returns something like

Name	Mark
Jones	85
Smith	65

The point being we did not instruct the SQL engine on how to find those results: it can choose any method it likes

Declarative and Imperative

It returns something like

Name	Mark
Jones	85
Smith	65

The point being we did not instruct the SQL engine on how to find those results: it can choose any method it likes

E.g., iterating through the table testing for `Course` then `Mark`

Declarative and Imperative

It returns something like

Name	Mark
Jones	85
Smith	65

The point being we did not instruct the SQL engine on how to find those results: it can choose any method it likes

E.g., iterating through the table testing for `Course` then `Mark`

Or, testing for `Mark` then `Course`

Declarative and Imperative

It returns something like

Name	Mark
Jones	85
Smith	65

The point being we did not instruct the SQL engine on how to find those results: it can choose any method it likes

E.g., iterating through the table testing for `Course` then `Mark`

Or, testing for `Mark` then `Course`

A sophisticated SQL engine will make a judgement and choose the most efficient search

Declarative and Imperative

> “Declarative” means you can use it without knowing what it’s doing. All too often, it means you can’t tell what it’s doing, either.

Anon