Computer Science Program, The University of Texas, Dallas

Classical MILs

Why MILs?

Main Concepts of MILs

Trend in Industry

Why Module Interconnection Languages (MILs)?

"Programming-in-the-small vs. programming-in-the-large"

	Programming-in-the-small	*Programming-in-the-small*
Goal	building "programs"	building "software"
Problem	Usually clear, small	Usually unclear, large
Emphasis	Detailed design & impl.	sw. architecture
Technique	Structured programming	"divide & conquer" "separation of concerns"
Notation	PLs	(Formal) (OO) Specification Lang.
Manpower	single person/small number	multi-person
Version	usually single	multi-version

Why Module Interconnection Languages (MILs)?

Programming-in-the-large requires conquering complexity!

✰ *Division of work*

Only the owner team needs to know how to implement a particular part

✰ *Multi-paradigm implementation*

Different people are good at different PLs
Different PLs are good for different things
Different things are developed at different times

✰ *Evolvable software*

Impact of changes should be localized
change in data structure or algorithm should be hidden
change in PLs should be hidden (or localized -> wrapper)

✰ *Information protection*

Only on need-to-know basis

✰ *Reuse of components (in the library)*

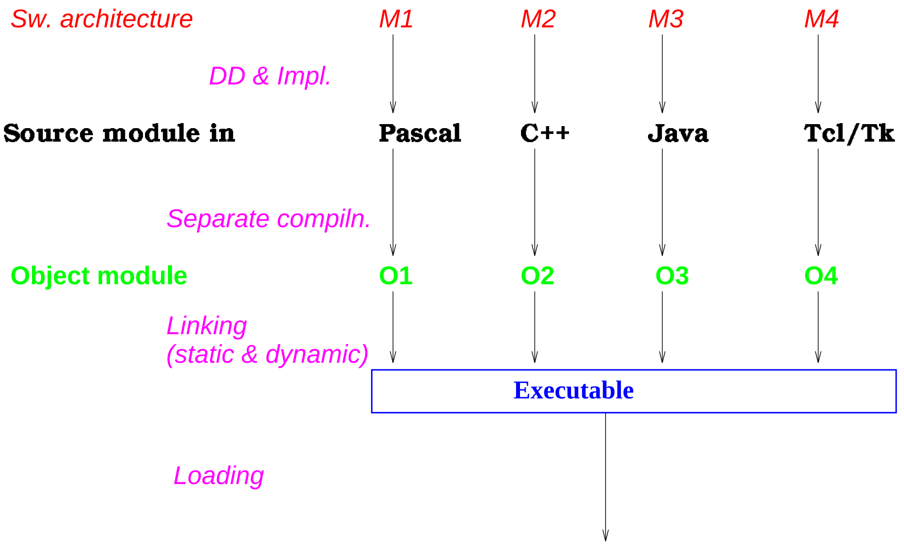
Reduce development & verification effort

✰ *Separate compilation*

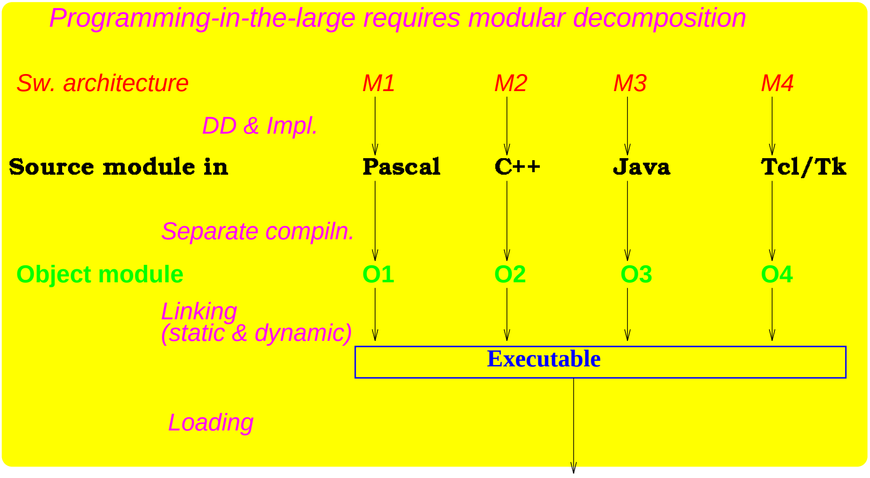
Can’t compile 1M LOC for each change

Why Module Interconnection Languages (MILs)?

Programming-in-the-large requires modular decomposition



Why Module Interconnection Languages (MILs)?



But

- ***static type-checking & consistency checking at an intermediate level of descr.***

  e.g., M1 uses a variable V in M3

  Is V defined in M3?
  Is M1 allowed to access V

- ***controlling different versions, assembling components for a complete system***

Main Concepts of MILs

MILs provide formal grammar constructs
for various module interconnection specifications
for assembling a complete software system.

- ***The first MIL was developed in 1975***
  ["Programming-in-the-Large versus Programming-in-the-Small",
  DeRemer & Kron, IEEE TSE 2(1), June, 1976]

- ***Variations among different schemes***
  ["Module Interconnection Languages",
  Prieto-Diaz & Neighbors, The Journal of Systems and Software 6, 1986]

- ***Module structure called "System Tree"***

  - modules that provide/export/synthesize resources and require/import/inherit them

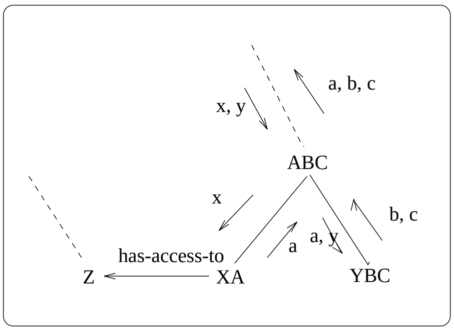
  - a resource is any entity that can be named in a PL
    (e.g., variables, constants, procedures, type defs)

  - interface-oriented,
    without details of how functions or modules are implemented

Main Concepts of MILs

- **Example**

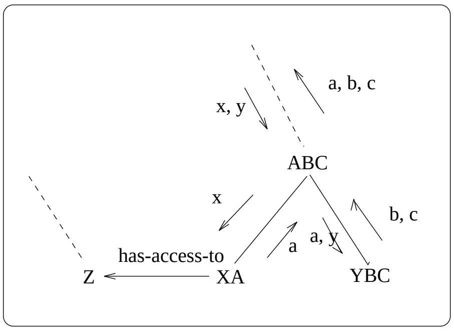
```
module ABC
     provides: a, b, c    /* resources defined in ABS - "statement of origin" */
     requires: x, y       /* resources used but not  in ABS - "statement of usage" */
     consist-of: functiona XA, module YBC /* nesting of module */

     function XA
           provides: a
           requires: x
           has-access-to: module Z  /* any recource provided by Z */
           real x, integer a
     end XA

     module YBC
           provides: b, c
           requires: a, y
           real y, integer a, b, c
      end YBC

end ABC
```



Main Concepts of MILs

- **Some constraints on accessibility**
  - has-access-to is not transitive

    e.g., L <------------ M <---------- N does not mean L <----- N
  - inheritance can be all (e.g., XA can access both x, y) or
    restricted (e.g., x only)
- **Questions**
  - If L <------ M and M’ is a child of M, can M’ access L?
  - If L <------ M and L’ is a child of L, can M access L’?



Main Concepts of MILs

☛ systems supporting module interconnection include:

✍ ***Ada (package), Module (module)***

highly modular and provide for version definitions

✍ ***Protel (PRocedure Oriented Type Enforcing Language)***

implemented in 1975 by BNR
used extensively but mainly by BNR
based on compile-link-load paradigm
performs type checking across modules

✍ ***SCCS (Source Code Control System)***

part of PWB (Programmer's Work Bench) facility
by Bell Labs in 1973
a file storage system for recording various versions of a text file
supports creation of any revision of a source program or text
file protection against accidental changes

✍ ***SARA (System ARchitect's Apprentice)***

supports a structural multi-level requirements-driven methodology
for the design of reliable sw or hw digital systems
developed at UCLA in 1976 and under continual development

Trend in Industry

✲ ***"buy, don't build"***

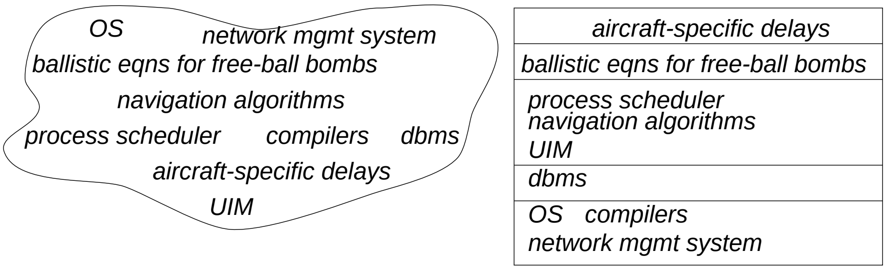
[Brooks, "No silver bullet: Essence and Accidents of Software Engineering", Computer 20(4), pp. 10-19, Apr. 1987]

faster (reduced development time)
increased reliability
increased flexibility

✲ ***increasing component size and complexity***

e.g1., TPS at GTE (1000 small, 50 large)

e.g2., F-22 fighter aircraft



OS
network mgmt system
ballistic eqns for free-ball bombs
navigation algorithms
process scheduler
compilers
dbms
aircraft-specific delays
UIM

aircraft-specific delays
ballistic eqns for free-ball bombs
process scheduler
navigation algorithms
UIM
dbms
OS compilers
network mgmt system

Trend in Industry

☼ ***coordination among components***

coordination infrastructure & standards for components plug-in

e.g., CORBA

☼ ***from subroutines to subsystems***

["Component-based Software Development",
American Programmer 8(11), Nov. 1995]

☼ ***architecture specification as a deliverable***

"If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development." [Boehm]

increasing importance of sw. architecture and specification