Component-Based Software
State-of-the-Art and Lessons Learned (part 2)

A. Richard Newton

Department of Electrical Engineering and Computer Sciences

University of California at Berkeley

Objects versus Components

"Object orientation has failed but component software is succeeding"
(Udell, 1994)

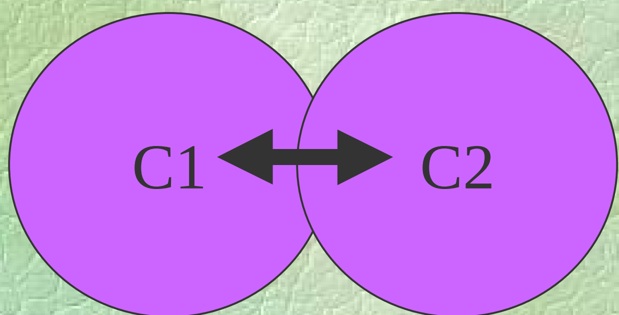
- Definition of objects is purely technical
  - Encapsulation of state and behavior, polymorphism, inheritance
  - Does not include notions of independence or late composition (although they can be added…)
- Object markets did not happen
  - Like the FPGA market-- vendors give the tools away to sell a companion product (e.g. MFC)
- In OO, construction and assembly share a common base
  - Development is very technical, assembly is very technical
  - In CO, construction is technical, but assembly must be open to a wider user base
- Objects are rarely shaped to support "plug-and-play"
- Typically a component has to have sufficiently many uses, and therefore clients, for it to be viable

Comments from the Microsoft TAB Meeting

- "Components don't work in software either"
  - Components only work if in a common family: testing of cross-products (e.g. visual basic)
  - Alternately, they need to have *very* simple and *very* standard I/O relationships (e.g. Unix pipes-character streams)
  - Or they need to be "large chunks of funtionality" (e.g. Oracle database)
- "The specification has to be much 'smaller' than the code"
- The ratio of glue required to size of component is also a critical issue
- Big issue is the testing/verification of the combinations

Component-Based Design

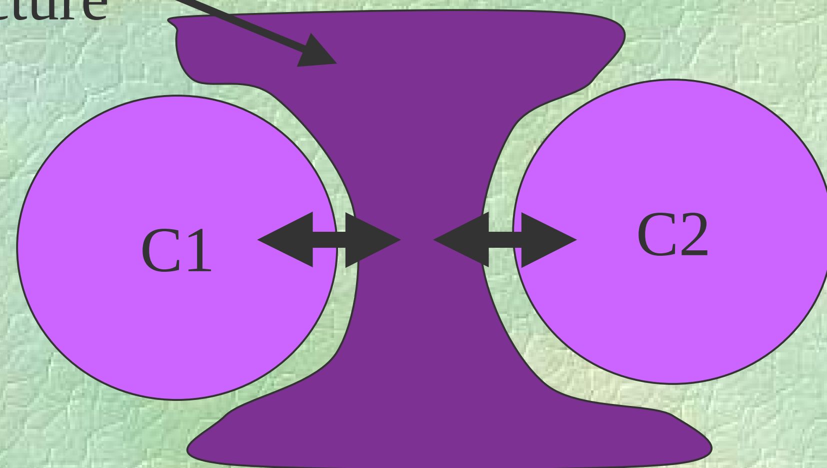
Today



"pass a pointer"
implemented on a stack

The component infrastructure

Tomorrow



Reliable, robust,
adaptable, and 'efficient':
"the operating system"

Component-Based Design

- In software languages:
  - Assume we are all on the same “team”
  - Optimize for efficiency, follow-up with debugging to fix problems (fragile interfaces)
  - Doesn’t scale well! (e.g. the Web)
- In communication protocols (e.g. TCP/IP)
  - Assume the guy at the other end is brain-dead
  - Assume whatever can go wrong will (links break, etc.)
  - Results in an “architecture” (e.g. packet-based) that is robust under the assumptions

Component-Based Design

- What is the "TCP/IP of component assembly"?
  - In the early days of TCP/IP we needed an IMP to implement the protocol, today it runs in s/w on a laptop
  - Must be reliable, robust, adaptable ("learn", self-optimizing, self-balancing, negotiate for resources…)
  - Self-verifying (what does that mean?)
  - Self-testing
  - "Queriable"
- In many ways, it's the "OS" of a component-oriented world
- Components might be collections of transistor, chunks of software (objects), applications, operating systems, NOW clusters, etc.

Next-Generation Operating Environments

- Advances in hardware and networking will enable an entirely new kind of operating system, which will raise the level of abstraction significantly for users and developers.
- Such systems will enforce extreme location transparency
  - Any code fragment runs anywhere
  - Any data object might live anywhere
  - System manages locality, replication, and migration of computation and data
- Self-configuring, self-monitoring, self-tuning, scaleable and secure

Adapted from Microsoft "Millenium" White Paper
http://www.research.microsoft.com

Next-Generation Operating Environments

- <u>Seamless Distribution</u>: System decides where computation should execute or data should reside, moving them there dynamically
- <u>Worldwide Scalability</u>:Logically there should only be one system, although at any one time it might be partitioned into many pieces.
- <u>Fault-Tolerance</u>: Transparently handle failures or removal of machines, network links, etc.

Next-Generation Operating Environments

- Self-Tuning: System should be able to reason about its computations and resources, allocating, replicating, and migrating computation and data to optimize performance, resource usage, and fault tolerance.
- Self-Configuring: New machines, network links, and resources should be automatically assimilated.
- Security: Allow non-hierarchical trust domains.
- Resource Controls: Both providers and consumers may explicitly manage the use of resources belonging to different trust domains.

Next-Generation Operating Environments

- No Storage Hierarchy: Once information is created, it should be accessible until it is no longer needed or referenced.
- Introspection: The system should posses some aspects of introspection and reflection.
  - Pervasively self-monitoring
  - Reason about its own configuration and performance
  - Suggest improvements
- Just-in-Time Binding: Sort of like the Internet today, but extended to all object interactions. “Binding-by-Search”
- Tools Emphasis Shifting: From code-efficiency to rapid application development with wizards automatically generating scaffolding or framework code.

"WebOS"

- The goal is to provide a common set of OS services to wide area applications, including mechanisms for:
  - Resource discovery
  - A global namespace
  - Remote process execution
  - Resource management
  - Authentication
  - Security
- Provide services needed to build applications that are:
  - Geographically distributed
  - Highly available
  - Incrementally scalable
  - Dynamically reconfiguring

Interfaces and Standards

- "A component needs to hold a significant portion of a market specific to its domain"
  - Generally drives (quasi) standards
- A standard should specific *just* as much about interfacing of certain components as is needed to allow sufficiently many clients and vendors to work together (including acceptable deviations and "tolerances")
- Wiring standards are not enough
  - People can find ways around wiring as needed: adaptors

Components and Interfaces

- Interfaces are the means by which components connect
  - "A set of named operations that can be invoked by clients"
  - Specification of the interface becomes the mediating middle that lets the two parties work together
- Direct (procedural) and indirect (object) interfaces
  - Object interface introduces an *indirection* called method dispatch
  - Has a big effect when versioning objects, for example
  - Very view solutions to this problem!

Interfaces as Contracts

- Not only requirements on the component, but also on the user, hence the term “contract” of “agreement”
- Best captured today by *preconditions* and *postconditions* (and perhaps *invariants*)
  - e.g. Eiffel (Meyer)
  - Hoare triple: {precondition} operation {postcondition}
- Non-functional requirements
  - It shouldn’t fail, it should recover, it shouldn’t take too long,…
- Example of layout checking as a component approach

Nonfunctional Requirements

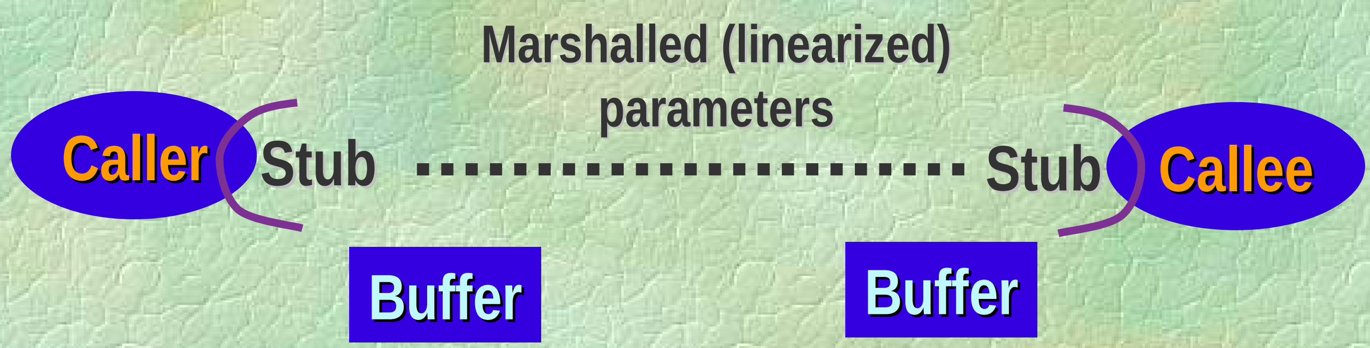
- Contracts usually state what is done under which provisions
- What about time taken, resources needed, etc?
  - Use of shared resources (e.g. heap)
  - In concurrent RT environment, priorities and their interactions
- Nonfunctional requirements can break components as well as functional ones
- C++ Standard Template Library (Usser & Saini, 96), execution time is bounded
  - Not in seconds, but as a complexity of legal implementation

Formal versus Informal

- Different parts of the interface can be specified more or less formally
- Formalizing wherever possible is a good idea: research
- Keep contracts as simple as possible
- Difficult when dealing with recursion and re-entrance
- Would like to have a compiler or tool check clients and providers for adherence to contracts

Interprocess Communication (IPC)

- Lots of ways:
  - files, pipes, sockets, semaphores, shared memory
  - all scale to networks, except shared memory
- All operate on level of bits and bytes
- Implementing complex operations on top of these mechanisms painful and error-prone
- RPC proposed in 1984 (Bird & Nelson)



OctObject Structure

```
struct octObject {
    octObjectType type;
    octId objectId;
    union {
        struct octFacet facet;
        struct octInstance instance;
        struct octProp prop;
        struct octTerm term;
        struct octNet net;
        struct octBox box;
        struct octPolygon polygon;
        struct octCircle circle;
        struct octPath path;
        struct octLabel label;
        struct octBag bag;
        struct octLayer layer;
        struct octPoint point;
        struct octEdge Edge;
        struct octChangeList changeList;
        struct octChangeRecord changeRecord;
    } contents;
};
```

The OctFacet Object

```
struct octFacet {                       /* facet object */
    char *cell;                         /* cellName */
    char *view;                         /* viewName */
    char *facet;                        /* "interface" or "contents" */
    char *version;                      /*  OCT_CURRENT_FACET */
    char *mode;                         /*  "r", ""w" or "a" */
};
```

The octPoint and octBox Objects

```
struct octPoint {                    /* oct Point */
    octCoord x;                      /* x coordinate (32-bit int )*/
    octCoord y;                      /* y coordinate (32-bit int) */
};
struct octBox {                      /* oct Box */
    struct octPoint lowerLeft;
    struct octPoint upperRight;
};
```

The octCircle Object

```
struct octCircle {                          /* oct Circle */
    octCoord startingAngle;                 /* times 1/10° */
    octCoord endingAngle;                   /* for slice */
    octCoord innerRadius;                   /* for donut */
    octCoord outerRadius;                   /* radius of circle */
    struct octPoint center;                 /* center point */
};
```

Operations on Facets

```
void octBegin()
void octEnd()

octStatus octOpenFacet(octObject *facet)
octStatus octCloseFacet(octObject *facet)

octStatus octOpenMaster(octObject *instance, *facet)
octStatus octOpenRelative(octObject *rfacet, *facet, int location)
octStatus octFlushFacet(octObject *facet)
octStatus octWriteFacet(octObject *new, *old)
octStatus octCopyFacet(octObject *new, *old)
octStatus octFreeFacet(octObject *facet)
octStatus octGetFacetInfo(octObject *facet, struct octFacetInfo *info)
octFullName(octObject *facet, char **name)
```

Operations on Data Items

```
octStatus octCreate(octObject *cnt, *obj)
octStatus octDelete(octObject *obj)
octStatus octModify(octObject *obj)

octStatus octAttach(octObject *cnt, *obj)
octStatus octDetach(octObject *cnt, *obj)

octStatus octAttachOnce(octObject *cnt, *obj)
octStatus octIsAttached(octObject *cnt, *obj)

octStatus octPutPoints(octObject *obj, int32 num, octPoint *pnts)
octStatus octGetPoints(octObject *obj, int32 *num, octPoint *pnts)
```

Retrieving Data Items

```
octStatus octInitGenContents(
    octObject *cnt, octObjectMask mask, octGenerator *gen)
octStatus octInitGenContainers(
    octObject *obj, octObjectMask mask, octGenerator *gen)
octStatus octGenerate(octGenerator *gen, octObject *obj)
```

Values for mask:

OCT_FACET_MASK	OCT_TERM_MASK
OCT_NET_MASK	OCT_INSTANCE_MASK
OCT_PROP_MASK	OCT_BAG_MASK
OCT_POLYGON_MASK	OCT_BOX_MASK
OCT_CIRCLE_MASK	OCT_PATH_MASK
OCT_LABEL_MASK	OCT_LAYER_MASK
OCT_POINT_MASK	OCT_EDGE_MASK
OCT_FORMAL_MASK	OCT_CHANGE_LIST_MASK
OCT_CHANGE_RECORD_MASK	

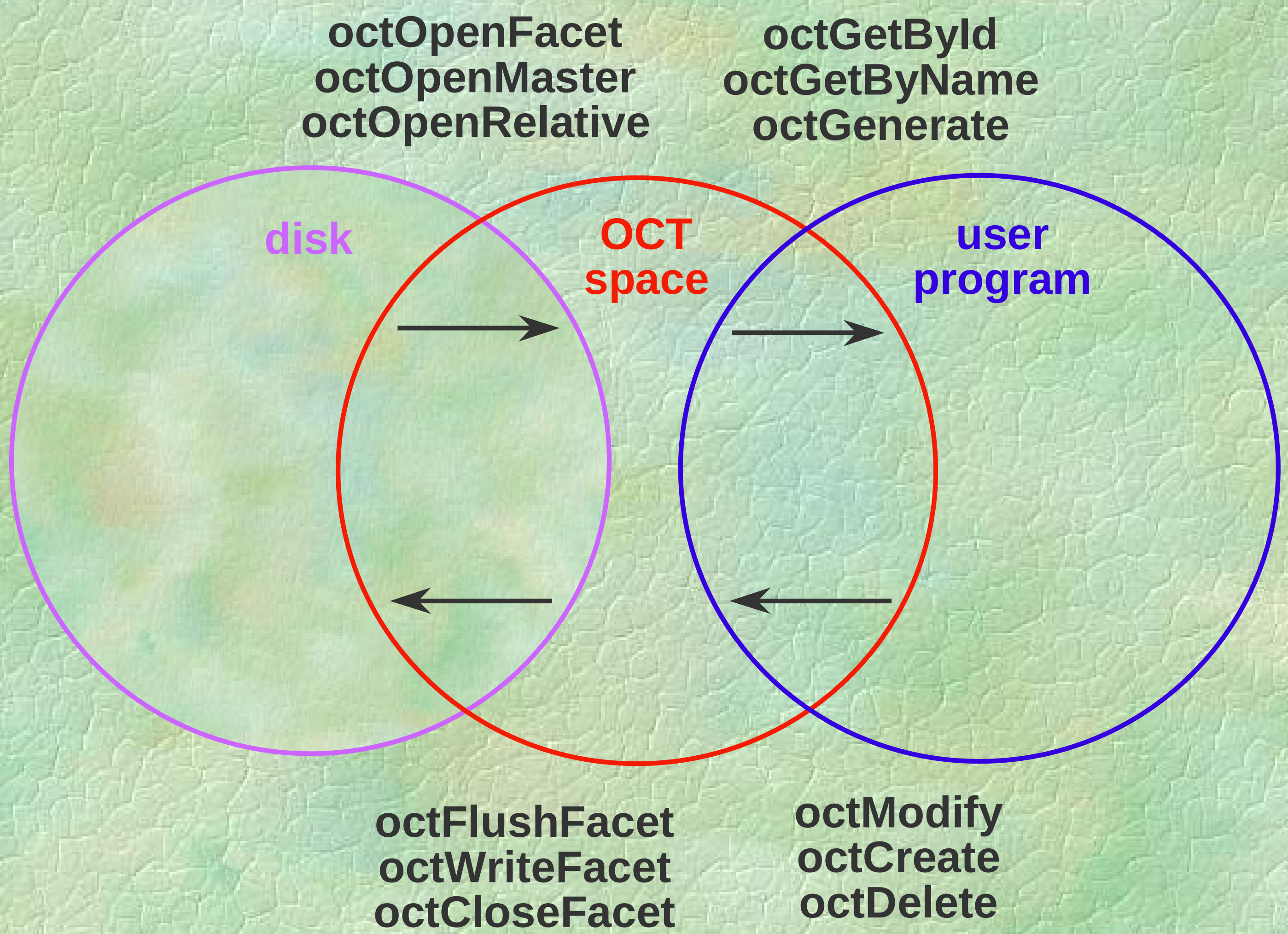
Use of Generators

```
/* proper way to generate */
    while (octGenerate(&gen, &obj) == OCT_OK) {
        /* do something */
    }

/* XXX wrong way to generate */
    while (octGenerate(&gen, &obj) !=
  OCT_GEN_DONE) {
        /* do something */
    }
```

Generator Examples

```
octInitGenContents(&facet, OCT_NET_MASK, &gen);
    while (octGenerate(&gen, &net) == OCT_OK) {
        /* do something */
    }
octInitGenContents(&facet, OCT_NET_MASK, &gen);
    while (octGenerate(&gen, &net) == OCT_OK) {
        octDelete(&net);
    }

/* XXX will loop infinitely */
newnet.type = OCT_NET;
newnet.contents.net.name = "new net";
octInitGenContents(&facet, OCT_NET_MASK, &gen);
    while (octGenerate(&gen, &net) == OCT_OK) {
        octCreate(&facet, &newnet);
    }
```

OCT Operations and the Environment



OCT Program Example

```
/*
 * program to generate over all geometries in the facet
 */
#include "copyright.h"
#include "port.h"
#include "oct.h"

main(argc, argv)
int argc;
char **argv;
{
    /* declare the oct objects to be used */
    octObject facet;       /* facet to be opened           */
    octObject layer;       /* layer containing the geometry */
    octObject geo;         /* geometry on the layer        */

    /* declare the oct generators to be used */
    octGenerator lgen;     /* generator for the layers     */
    octGenerator ggen;     /* generator for the geometries  */

    /* initialize oct - allocate tables, notify design managers, etc. */
    octBegin();
```

OCT Program Example

```
/*
 * open the facet
 */
facet.type = OCT_FACET;
facet.contents.facet.cell = argv[1];
facet.contents.facet.view = argv[2];
facet.contents.facet.facet = "contents";
facet.contents.facet.version =
 OCT_CURRENT_VERSION;
facet.contents.facet.mode = "r";

if (octOpenFacet(&facet) < OCT_OK) {
    octError("opening facet to be generated");
    exit(-1);
}
```

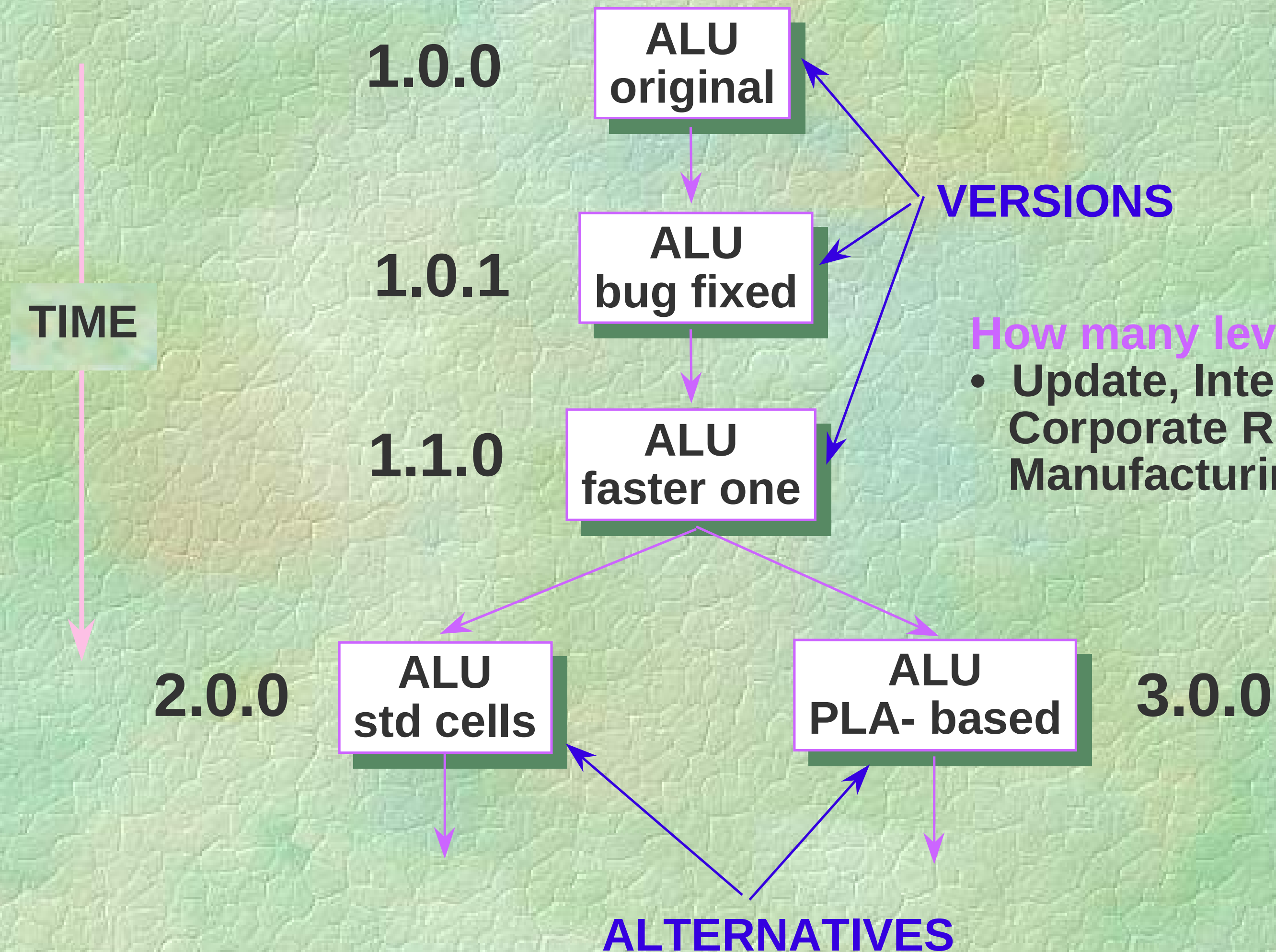
OCT Program Example

```
/*
 * generate over all layers
 */
    (void) octInitGenContents(&facet, OCT_LAYER_MASK, &lgen);
    while (octGenerate(&igen, &layer) == OCT_OK) {

        /*
         *  generate over all geometries on the layer
         */
        (void) octInitGenContents(&layer, OCT_GEO_MASK, &ggen);
        while (octGenerate(&ggen, &geo) == OCT_OK) {

              /*
            * process the geometry
            */
        }
    }
  /* close down oct - release memory, notify design managers, etc. */
    octEnd();
    exit(0);
}
```

OCT Example Program

```
/*
 * generate over all layers
 */

(void) octInitGenContents(&facet, OCT_LAYER_MASK, &lgen);
while (octGenerate(&igen, &layer) == OCT_OK) {

    /*
     *  generate over all geometries on the layer
     */

    (void) octInitGenContents(&layer, OCT_GEO_MASK, &ggen);
    while (octGenerate(&ggen, &geo) == OCT_OK) {

        /*
         * process the geometry
         */
    }
}
```

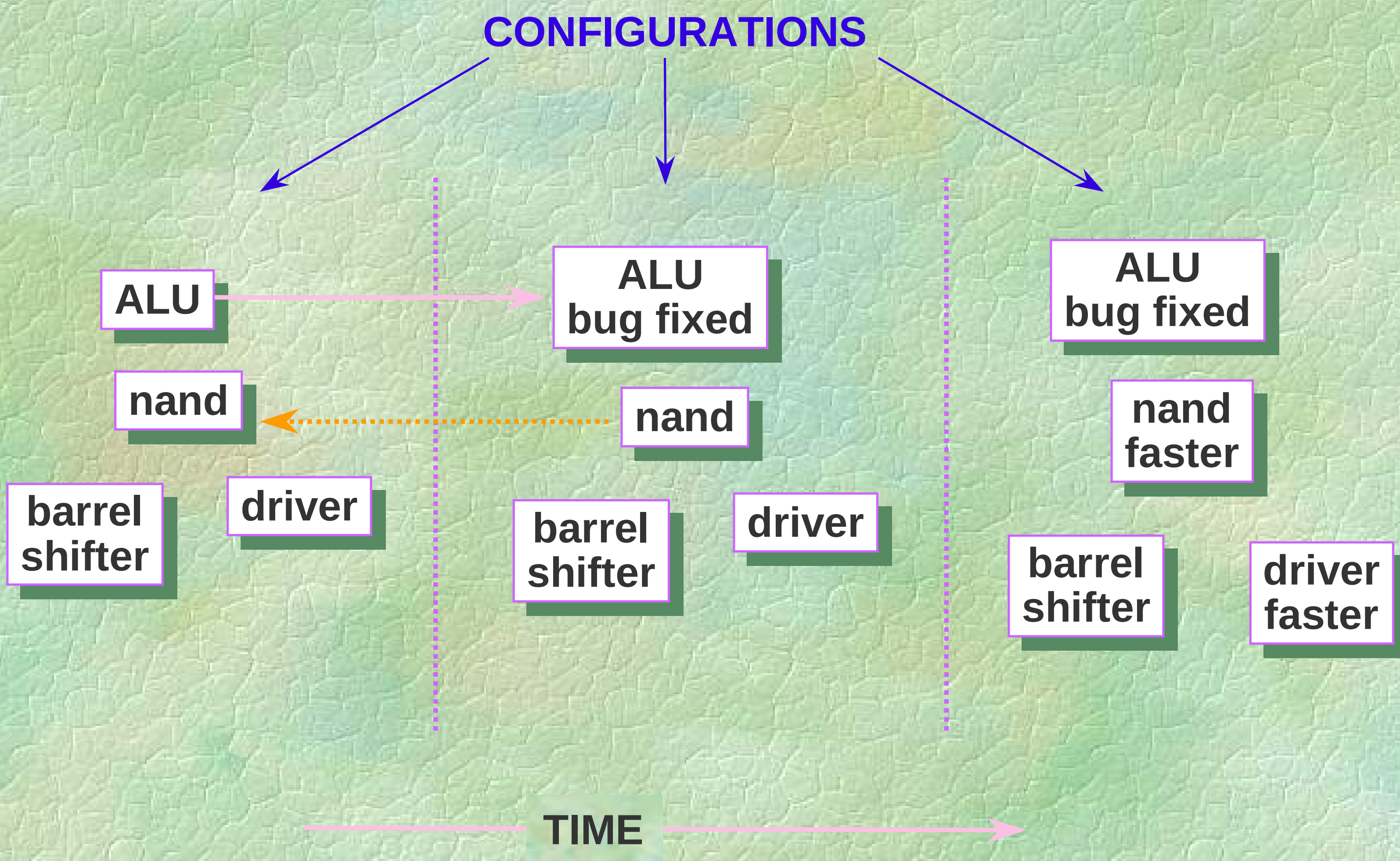
Versions, Alternatives, and Configurations



How many levels?

- Update, Internal Release, Corporate Release, Manufacturing Release, ...

Versions, Alternatives, and Configurations



Some Potential Key Technologies

- What software technology, or technologies, will play the central role in enabling such a distributed component architecture?

- Java and *JavaBeans*
- CORBA
- Microsoft COM (COM, DCOM, COM+)
- Jini

CORBA
(Common Object Request Broker Architecture)

- A standard for distributed objects being developed by the Object Management Group (OMG).
- CORBA provides the mechanisms by which objects transparently make requests and receive responses, as defined by OMG's ORB.
- The CORBA ORB is an application framework that provides interoperability between objects, built in (possibly) different languages, running on (possibly) different machines in heterogeneous distributed environments.

CORBA (1.0 1991, 2.0 1995)

- Very open approach: a “wiring” model
- Connects a wide variety of languages, implementations, and platforms
- CORBA components cannot operate on an efficient binary level, but must engage in expensive high-level protocols
  - e.g. Internet Inter-ORB protocol (IIOP)
  - Visigenic ORB “Visibroker”, part of Netscape browser
- Object interface described in a common interface definition language (IDL)
  - All languages must have bindings to OMG IDL

CORBA

Invocation Interface | ORB Interface .... ORB Interface | Object Adapter

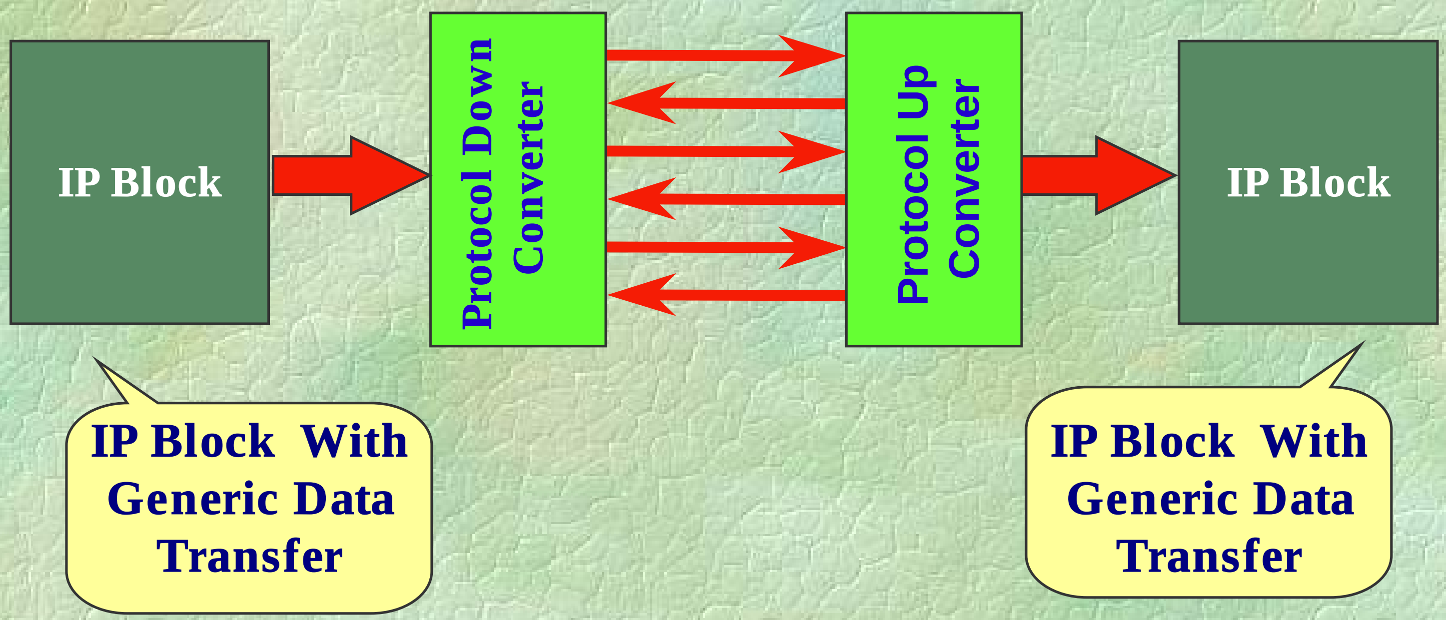
obj.m(args)

```
Module Example {
  struct Date {
    unsigned short Day;
    unsigned short Month;
    unsigned short Year;
  }
  interface Ufo {
    readonly attribute unsigned long ID;
    readonly attribute string Name;
    readonly attribute Date FirstContact;
    unsigned long Contacts ();
    void RegisterContact (Date dateOfContacts);
  }
}
```

Object
method1
...
method n

Communication Refinement

- Separate *Function* of blocks from inter-block *Communication*
- Substitute lower-level detail for communications behavior



Source: Prof. Alberto Sangiovanni

Communication Refinement

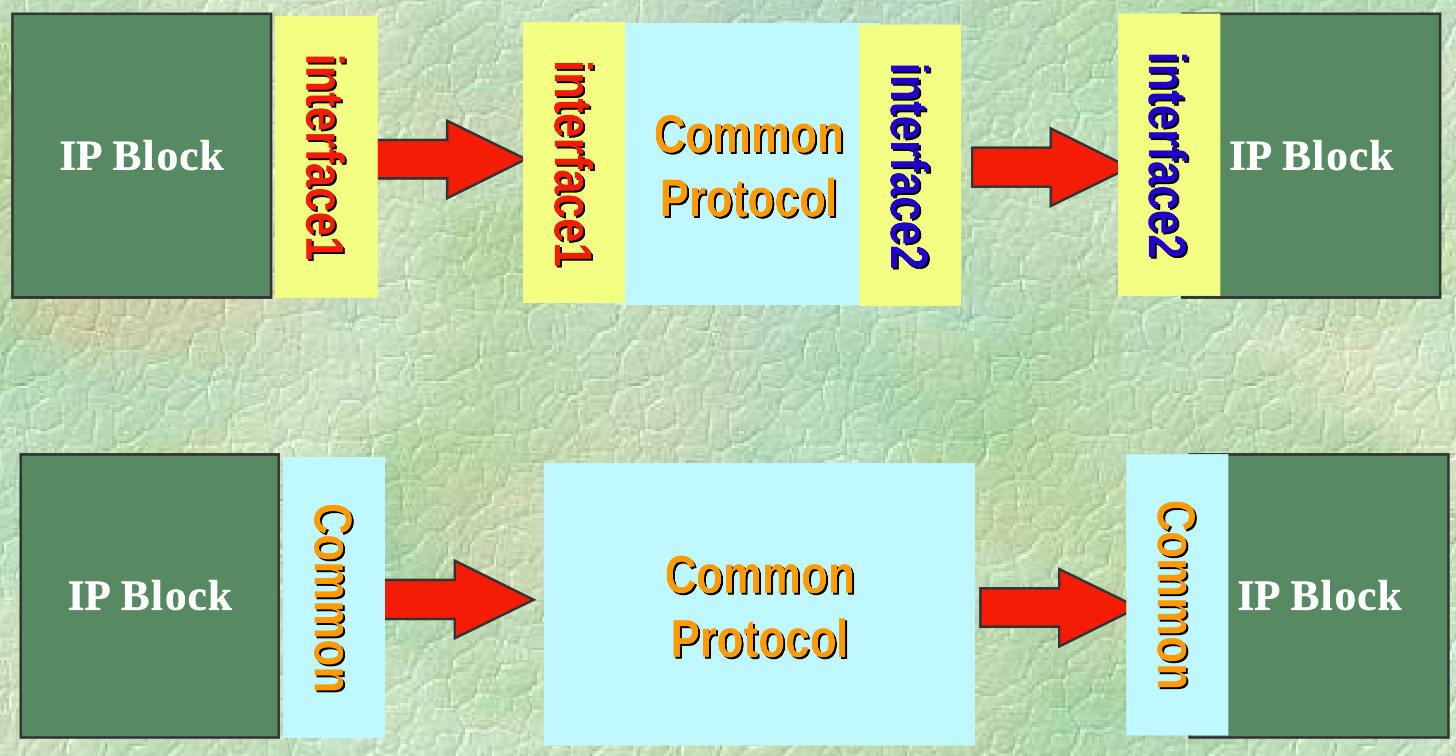
- Issue: Where do we cut? Where are the “standards”?
- Where is the communication burden placed?
- Applies to both hardware and software

Microsoft COM Analogy (Component Object Model)

- Binary and network (DCOM) standard that allows two objects to communicate, regardless of what machine they are running on.
- Can be used from C++, C, VB, Java, Delphi, …
- Supports three types of objects: In-process (DLL), local (EXE), and remote (DLL or EXE)

Communication Refinement

- Issue: Where do we cut? Where are the “standards”?
- Where is the communication burden placed?
- Applies to both hardware and software



Java/JavaBeans Analogy

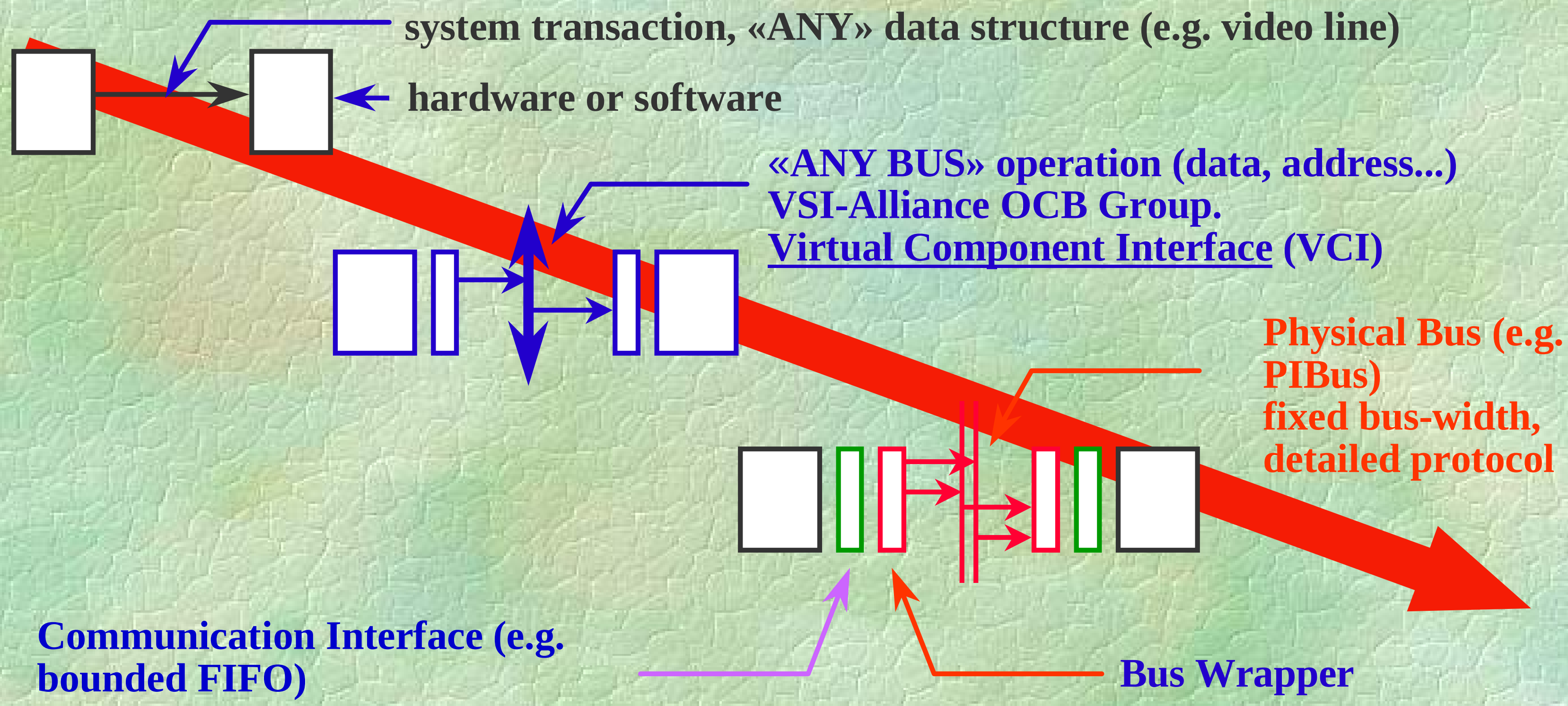
- *JavaBeans* is a portable, platform-independent component model written in Java.
- It enables developers to write reusable components once and run them anywhere -- benefiting from the platform-independent power of Java.
- *JavaBeans* acts as a bridge between proprietary component models and provides a seamless means for developers to build components that run in ActiveX container applications.

Attributes of *JavaBeans*

- Introspection: enables a builder tool to analyze how a Bean works
- Customization: enables a developer to use an app builder tool to customize the appearance and behavior of a Bean
- Events: enables Beans to communicate and connect together
- Properties: enable developers to customize and program with Beans
- Persistence: enables developers to customize Beans in an app builder, and then retrieve those Beans, with customized features intact, for future use

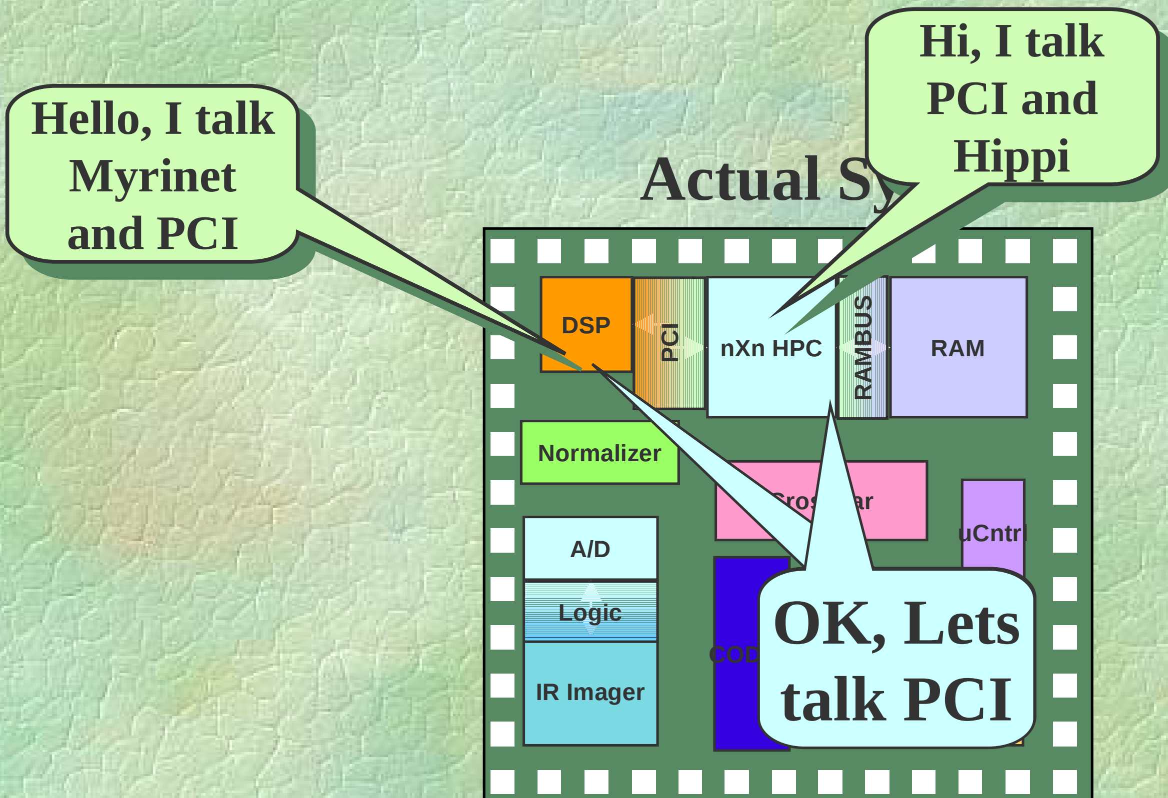
Communication Refinement

Standard interfaces constitute the backbone of an IP market: abstract form the concerns of hardware implementation (multi-target VC), abstract from the concerns of a particular bus (bus-independent VC)



Source: Prof. Alberto Sangiovanni

Automated Interface Synthesis



Source: DARPA ISAT *Silicon 2010* Study, 1997 (Randy Harr, Synopsys)