What's Special About "embedded" Systems?

A. Not just a question of size or real-time demands

B. Definition--an embedded computer system is considered a module in some larger system

C. Some distinguishing characteristics of embedded computer systems

   1. Designer not free to define interface.
   2. Interface constraints may be strict and arbitrary, but we can't ignore them.
   3. Several similar interfaces may be involved.
   4. Interface will change during development
   5. Cost of changing computer system often not considered seriously when changes in total system are made.

A Contractual Dilemma (internal or external)

1. Contract must constrain contractor by providing testable specifications.
2. For many reasons, final interface must be considered unknown.
3. System for "wrong" interface will be hard to change.
4. Lack of competition makes late changes unreasonably expensive.

These dilemmas arise in any multi-person project - they are simply exacerbated when contracts are involved.

Preponderance of Embedded Systems

This is often used by U.S. DoD as an explanation for the high cost of its software.

- a partial explanation for the high cost of DoD software
- Even more of a problem in communication systems.

1. Real-time constraints, limited memory, etc. are not unique to DoD
2. Incompetence is not confined to DoD
3. Poor tools can be found everywhere
4. Language proliferation can be found elsewhere
5. Rigidity and complexity of interfaces is a characteristic of several application areas but Communications and DoD are extreme examples.

Examples of Embedded Systems

A. Mailing List System

Constraints: address format,
use of titles,
postal code scheme,
location of key information

B. Message Processing Systems

Constraints: Message format,
priority scheme,
addressing scheme

C. Computers in weapons systems

(e.g., TC-2 computer in the A-7)

Constraints: devices,
navigation data,
weapon characteristics

D. Telephone Switching Systems

Constraints: Other company's switches
Own older switches
international standards
telephone number rules

What is an Interface?

Before we can talk about design and specification of interfaces, we must define that term.

More than just syntax or format.

The interface between two programs is characterised by the set of explicit and implicit assumptions they make about each other.

Examples of implicit assumptions and their effects on application programs

Mailing List Software

- all digit zip code
- six digit zip code
- zip code at end of address

A-7

- device data formats
- symbols on HUD
- data entry protocol

Applying "Information Hiding" to the "Embedded System" Problem

The external interface is what is likely to change.

Use an "abstract interface" to "hide" the actual interface.

Abstract Interfaces

What do we mean by abstract?

1. Do not mean vague, theoretical or highly mathematical; abstract means - expressing a general property.
2. Abstract implies a many-to-one mapping.
3. The abstraction represents many things equally well.
4. The abstraction models some aspects of the real things, but not all.
5. Eliminating detail is the approach: The interesting issue is *which* details should be eliminated.
6. Examples of abstractions
   - 6.1 Circuit diagrams
   - 6.2 Graphs
   - 6.3 Algorithms
   - 6.4 Data types

Abstract Interfaces

Why are abstractions useful?

1. If all properties of the abstract system correspond to properties of the real system, we can learn about the real system by studying the abstraction.

2. Abstraction is simpler (in principle).

3. Results about abstraction may be "reused".

Abstract Interfaces

What is an abstract interface?

1. One interface that represents many possible actual interfaces.

2. An interface that models some properties of actual interface but not all.

3. A subset of the set of assumptions in the actual interface.

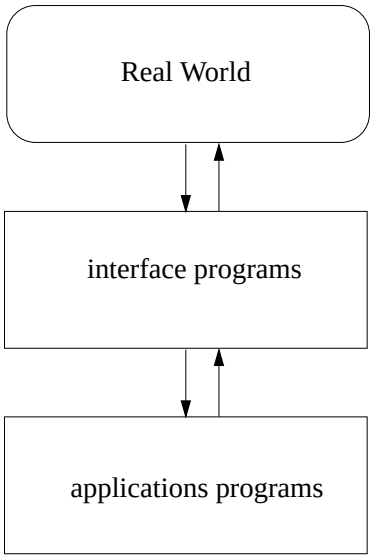
The assumptions must be true. *A lie is not an abstraction.*

All things true of the abstract interface must be true of actual interface.

How can interface hiding modules be used in embedded systems?

A. Define the abstract real-world interface.

B. Procure applications programs based on abstract interface, preventing exploitation of facts that happen to be true of today's actual interface.



actual real-world interface

abstract interface module

abstract real-world interface

The interface programs implement one instance of the many-to-one mapping between the actual real-world interface and the abstract real-world interface.

Cont'd. ⟶

How can interface hiding modules be used in embedded systems?

C. When actual interface is fixed, build interface programs.

D. "Real-World" changes that affect actual interface SHOULD only affect the interface programs.

E. There should be no unnecessary effort in translation from an external format to an internal one.

Simple Example--A Date Interface

Possible formats in actual interfaces:

February 10, 1941	(month day-in-month, year),
10 February 1941	(day-in-month month year),
10 February 41	(day-in-month month last-two-digits-of year)
10.2.1941	(day-in-month.integer-encoded-month.year),
2/10/1941	(integer-encoded month/day-in-month/year),
41.2.10	(last-two-digits-of-year.integer-encoded-month.day-in-month),
41 February 10	(last-two-digits-of-year month day-in-month)
41,41	(day-in-year, last-two-digits-of-year).
41,41	(last-two-digits-of-year, day-in year).
15015	(days since the first day of 1900)

Abstract Interface

get_year,get_month,get_day,-get-days-since-1900,
get-day-in-year, get-years-since-1900,
get-month-as-integer, etc.

Note assumptions about range of years.

How to Design an Abstract Interface

A. Prepare a list of assumptions about properties of all the possible real-world interfaces to be encountered--review it.

B. Express these assumptions by defining a set of access programs representing possible system inputs and outputs and by stating relations between these access programs. This is the module specification.

C. Perform consistency checks
   1. Verify that any property of the access program set is implied by the assumptions.
   2. Verify that all assumptions are in the interface specification.
   3. It should be possible to write bulk of system in terms of these access programs; if not, return to A.

D. Contracting:
   Contractor is required to write bulk of system in terms of the access programs defined in B, and programs must be correct for any implementation of those access programs that satisfies the specification.

   A separate contract is let for the Interface Modules.

Illustration of this Procedure for the Address Holder

1. Initial assumptions

The following items of information will be contained in addresses and can be identified by analysis of the input data; this information is the only information that will be relevant for our computer programs:

Last name

First name

Organisation

Street address

City, state and zip code

(single line with a comma between city and state)

Illustration of this Procedure for the Address Holder

2. Objections

There might be a title, e.g. Professor, King.

The postal code might be on a separate line.

There might be a P.O. Box.

It might be the name of a company,

. . . .

3. Refined assumptions - based on objections.

Many additional access programs are needed.

Refining/extending the interface for a subset of the interfaces

A. Some useful applications programs may not be generally applicable.

B. Confinement of the specialised program.

C. Specialisation (refinement) by adding access programs not generally implementable.

D. Specialisation (refinement) by stating additional properties of access programs.

E. Deviant actual interfaces.

F. The family tree again.

When won't it work?

A. Success depends on our ability to predict change (oracle assumption).

B. Success depends on existence of commonality between actual interfaces (interface programs smaller than applications programs).

C. The Big "Big-Box" Assumption.

Summary

A. An interface is equivalent to a set of assumptions.

B. The abstract interface is a precise, formally specified interface.

C. The abstract interface is a model of all "expected" actual interfaces. Explicit, systematic design is needed. Review is essential.

D. Contractor is more tightly constrained than in conventional procedure--his program is not allowed to make assumptions that limit applicability.

E. Actual interface is met by writing additional programs--not by modifying programs that were written based on the abstract interface definitions.

Abstract interface design as an application of fundamental principles

A. Being explicit about assumptions and design decisions.

B. Encapsulation of likely change.

C. Abstract interface modules can solve the embedded computer system problem by hiding the embedding from the computer.

D. External interface modules are just a special case--use same method for other information-hiding modules.