UNIVERSITY OF TECHNOLOGY SYDNEY

# SOFTWARE ARCHITECTURE FOR PRODUCT LINES

Tom McBride

UTS: ENGINEERING AND INFORMATION TECHNOLOGY

## Contents

UNIVERSITY OF TECHNOLOGY SYDNEY

Definition and examples of product lines

## WHAT ARE PRODUCT LINES

## A product line defined

- A product line is;
  - a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.
- Product lines reuse intellectual assets in multiple projects or products.

## What are product lines

- Families of products
- Share most of their essential functions
- Adapted to different needs through selecting and modifying features

## Product lines are common

- Engine, transmission, brakes, suspension, entertainment, air conditioning are all present but all can differ across models
- Connections between these components is likely to be the same.
- Chassis is largely the same
- Body shell changes for different models

## Camera product lines

- Common functions: zoom, focus, take photo, store image, manage images, power management, more…
- Common components: Image processor, sensor
- Different: pixels, lens size, body, battery, flash

## Software product lines

- ERP systems
- CRM systems
- Defect tracking systems
- Most packaged software would qualify as a product line

## Summary

- Similar software systems created from a shared set of software assets using a common means to deliver functions and features.

- Borrowed from manufacturing where the same assembly line is used to assemble similar products using similar components.

UNIVERSITY OF TECHNOLOGY SYDNEY

## ADVANTAGES OF PRODUCT LINE ARCHITECTURES

## A product line offers considerable advantages

- The improvements in cost, time to market, and productivity that come with a successful product line can be breathtaking. Consider:
  - Cummins, Inc., was able to reduce the time it takes to produce the software for a diesel engine from about a year to about a week.
  - Motorola observed a 400% productivity improvement in a family of one-way pagers.
  - Hewlett-Packard reported a time to market reduced by a factor of seven and a productivity increase of a factor of six in a family of printer systems.
  - With a family of satellite ground control systems it commissioned, the U.S. National Reconnaissance Office reports the first product requiring 10% the expected number of developers and having 90% fewer defects.

## What makes product lines work?

- The essence of a software product line is the disciplined, strategic re-use of assets in producing a family of products. What makes product lines succeed so spectacularly from the vendor or developer's point of view is that the commonalities shared by the products can be exploited through re-use to achieve production economies. The potential for re-use is broad and far-ranging, including:
  - Requirements.
    - Most of the requirements are common with those of earlier systems and so can be re-used. Requirements analysis is saved.
  - Architectural design.
    - An architecture for a software system represents a large investment of time from the organization's most talented engineers. As we have seen, the quality goals for a system—performance, reliability, modifiability, and so forth—are largely allowed or precluded once the architecture is in place. If the architecture is wrong, the system cannot be saved. For a new product, however, this most important design step is already done and need not be repeated.

## More on what makes product lines work

- Elements.
  - Software elements are applicable across individual products. Far and above mere code re-use, element re-use includes the (often difficult) initial design work. Design successes are captured and re-used; design dead ends are avoided, not repeated. This includes design of the element's interface, its documentation, its test plans and procedures, and any models (such as performance models) used to predict or measure its behaviour. One re-usable set of elements is the system's user interface, which represents an enormous and vital set of design decisions.
- Modeling and analysis.
  - Performance models, schedulability analysis, distributed system issues (such as proving absence of deadlock), allocation of processes to processors, fault tolerance schemes, and network load policies all carry over from product to product.
  - CelsiusTech reports that one of the major headaches associated with the real-time distributed systems it builds has all but vanished.
  - When fielding a new product in the product line, it has extremely high confidence that the timing problems have been worked out and that the bugs associated with distributed computing—synchronization, network loading, deadlock—have been eliminated.

- Testing.
  - Test plans, test processes, test cases, test data, test harnesses, and the communication paths required to report and fix problems are already in place.
- Project planning.
  - Budgeting and scheduling are more predictable because experience is a high-fidelity indicator of future performance. Work breakdown structures need not be invented each time. Teams, team size, and team composition are all easily determined.
- Processes, methods, and tools.
  - Configuration control procedures and facilities, documentation plans and approval processes, tool environments, system generation and distribution procedures, coding standards, and many other day-to-day engineering support activities can all be carried over from product to product. The overall software development process is in place and has been used before.

## More on what makes product lines work

- People.
  - Because of the commonality of applications, personnel can be fluidly transferred among projects as required. Their expertise is applicable across the entire line.
- Exemplar systems.
  - Deployed products serve as high-quality demonstration prototypes as well as high-quality engineering models of performance, security, safety, and reliability.
- Defect elimination.
  - Product lines enhance quality because each new system takes advantage of the defect elimination in its forebears. Developer and customer confidence both rise with each new instantiation.
  - The more complicated the system, the higher the payoff for solving vexing performance, distribution, reliability, and other engineering issues once for the entire family.

## Costs of developing a product line

- The first few cost more
  - All the costs of developing fresh software
  - Add the costs of designing for general use
- Start recovering costs after the third product in the same line
- Higher maintenance costs
  - More constraints on possible solutions

## How product lines can fail

- Developing a product line incurs extra costs to ensure generality

- These factors can contribute to product line failure:
  - Lack of a champion in a position of sufficient control and visibility
  - Failure of management to provide sustained and unwavering support
  - Reluctance of middle managers to relinquish autocratic control of projects
  - Failure to clearly identify business goals for adopting the product line approach
  - Abandoning the approach at the first sign of difficulty
  - Failure to adequately train staff in the approach and failure to explain or justify the change adequately

## Summary

- The costs of developing a product line architecture are immediate, the benefits are in the future
- ROI depends on variation – more variation reduces the potential return
- Original development costs are amortised over multiple instances of the product
- Product line development requires organizational commitment
- Customers can only be offered features and functions that the product line supports

## DEVELOPING A PRODUCT LINE ARCHITECTURE

## So, what does it take to start

- A little forethought and planning
- Do you expect to develop more than one?
  - How can you reduce the effort and cost of developing #2, #3, #4
- Best if there is some demonstrable need for it
  - New line of related products
  - Need to rationalise a collection of related products

## Shifting from craft to production

- Craft production can deliver exactly what the customer wants.
- Mass production can deliver most of what the customer wants much faster and at much lower cost.

## Product and Production Processes

- To build a product line it is necessary to build both the product and the system of production.
- Bespoke systems can be developed differently each time.
- Product line systems need a consistent method of development and production for consistent results.
- A consistent development method reduces flexibility and responsiveness while increasing quality and reducing cost

## Architectures for product lines

- Of all of the assets in a core asset repository, the software architecture plays the most central role.
- The essence of building a successful software product line is discriminating between what is expected to remain constant across all family members and what is expected to vary.
- Product line architecture concerns itself with a set of explicitly allowed variations, whereas with a conventional architecture almost any instance will do as long as the (single) system's behavioural and quality goals are met.
- A product line architect needs to consider three things:
  - Identifying variation points
  - Supporting variation points
  - Evaluating the architecture for product line suitability

## Start with the business processes

- Systems have similar high level business processes
  - Supply Chain Operations Reference Model (SCOR) – Source, Make, Deliver, Return
  - University
    - Student management: Enrol students, assess students, record results, graduate students
    - Course management: Develop courses, develop subjects, teach subjects
  - General commerce
    - Client management: Register, teach, assess
    - Event management: Schedule, recruit participants, conduct
    - Inventory management: Acquire, maintain, dispose, lend, return

## Identifying variation points

- How does one organization differ from another
- How do business processes vary between organizations
- What is likely to change
  - Technology
  - Scale
  - Peripheral components
  - Performance requirements
  - Languages and other user interface factors
- What is hard to change
  - Workflow
  - Interfaces and infrastructure
  - Limits, whether anticipated or accidental

# Supporting variation points

- In software product lines, variation can be achieved by
  - Parameterization:
    - Parameterization is a variability mechanism that allows a reuser to change the values of the attributes in a core asset component. This can be done by providing the capability through the component's interface to initialize or change the value of parameterized attributes.
  - Information hiding:
    - Modeling variability using information hiding is where several versions of the same component are built with the same interface. The variability is hidden inside each version of the component. In this case, the variants are the different versions of the same component. However all variants of a component must adhere to the same interface. This approach works well as long as changes can be limited to individual components and no changes are required to the interface.
  - Inheritance
    - Modeling variability using inheritance is where variants are provided that do not have to adhere to the same interface. Variants can be specializations of other components where a subclass can extend the interface provided by a superclass by adding new operations or over-riding existing operations.

## What makes software product line difficult?

- Resistance to adopting a product line approach within the organisation
- Support tools like configuration management need to support greater complexity
- Difficult to convince customers they can’t have everything they want

# Product lines evolve

- Product line evolution must be managed in order to avoid chaos
  - Core assets must evolve, if possible (We learn how to do it better)
  - Evolution must be controlled to avoid feature bloat, mismatched components, blind alleys
- Customers learn the system and start demanding more variations
- Eventually the system encounters its architectural limits
  - Database size
  - Performance limits
  - Competing variations

## The product line mindset – mastering abstraction

- Successful product line developers are concerned about developing the *product line*
- Unsuccessful product line developers are more concerned about developing a lot of *products*
- Successful product line developers learn to use the abstracted features of the product line to develop new, but related, business

## Summary

- Start with the business processes. Determine actual and possible variations in the business process
- Variation is supported by three main techniques
  - parameterisation,
  - information hiding,
  - inheritance
- Architecture evaluation becomes more demanding
- Product lines must be managed to avoid chaos of feature bloat, mismatched components and blind alleys
- Consider how the product might evolve
- Consider the effect of changing technology

## CASE STUDY – CELCIUSTECH

## CelciusTech build control systems

- CelsiusTech has long been known as a leading supplier of command-and-control systems within Sweden's largest, and one of Europe's leading, defense industry groups, which also includes Bofors, Kockums, FFV Aerotech, and Telub.
- At the time they developed the systems, CelsiusTech was composed of three companies:
  - CelsiusTech Systems (advanced software systems),
  - CelsiusTech Electronics (defense electronics), and
  - CelsiusTech IT (information technology systems).
- It employed approximately 2,000 people and had annual sales of 300 million U.S. dollars. Their main site is near Stockholm, with subsidiaries located in Singapore, New Zealand, and Australia.

## The Ship System 2000

- CelsiusTech's naval product line, known as Ship System 2000 (internally as Mk3), provides an integrated system that unifies all weapons, command-and-control, and communication systems on a warship. Typical system configurations include 1 million to 1.5 million lines of Ada code distributed on a local area network (LAN) with 30 to 70 microprocessors.
- A wide variety of naval systems, both surface and submarine, have been or are being built from the same product line. These include the weapons, command-and-control, and communications portions of the following:
  - Swedish Göteborg class coastal corvettes (KKV) (380 tons)
  - Danish SF300 class multi-role patrol vessels (300 tons)
  - Finnish Rauma class fast attack craft (FAC) (200 tons)
  - Australian/New Zealand ANZAC frigates (3,225 tons)
  - Danish Thetis class ocean patrol vessels (2,700 tons)
  - Swedish Gotland class A19 submarines (1,330 tons)
  - Pakistani Type 21 class frigates
  - Republic of Oman patrol vessels
  - Danish Niels Juel class corvettes
- The Naval Systems division has sold more than 50 of its Mk3 naval systems to seven countries.

# The multi-role Corvette

## Some of the variable requirements

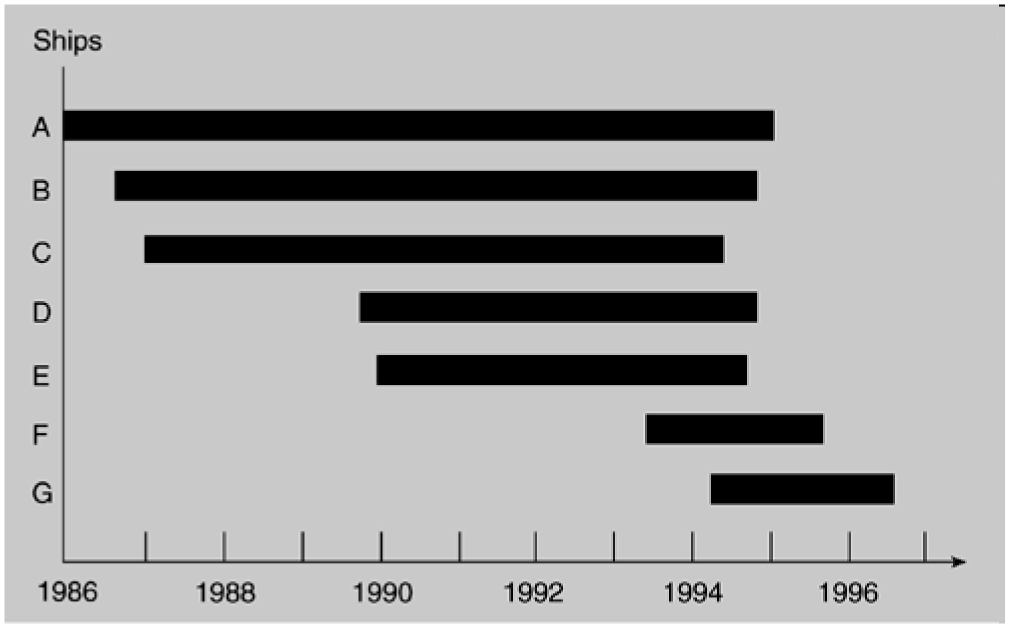
- Systems built from the product line vary greatly in size, function, and armaments.
- Each country requires its own operator displays on different hardware and in different presentation languages.
- Sensors and weapons systems, and their interfaces to the software, also vary.
- Submarines have different requirements than surface vessels.

## Variable requirements, continued

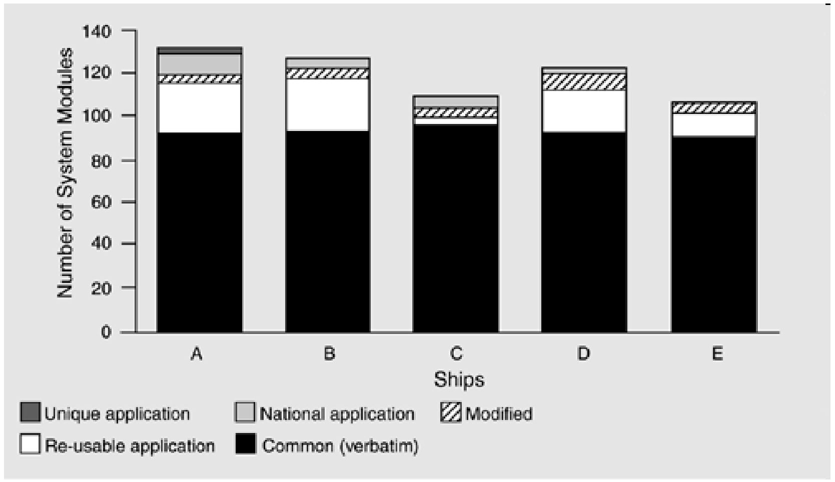
- Computers in the product line include
  - 68020,
  - 68040,
  - RS/6000, and
  - DEC Alpha platforms.
- Operating systems include
  - OS2000 (a CelsiusTech product),
  - IBM's AIX,
  - POSIX,
  - Digital's Ultrix, and others.
- The SS2000 product line supports this range of possible systems through a single architecture, a single core asset base, and a single organization.

# Business drivers – Shrinking development times

- Ships A & B contracted at the same time
- A & B took 9 years
- C & D started later but took less time
- E & F dramatically shorter development time due to extensive reuse



# Business drivers – code reuse



- 70% to 80% of the components are re-used without modification

## Using core assets to expand business areas

- CelsiusTech has expanded its business into a related area that takes advantage of the architecture and other core assets that were originally developed for naval uses.
- STRIC, a new air defense system for the Swedish Air Force, embraces the abstraction that a ground platform is a ship whose location does not change very often and whose pitch and roll are constantly zero.
- Because of the flexibility (ease of change) of the SS2000 architecture and product line, CelsiusTech was able to quickly build STRIC, lifting 40% of its elements directly from the SS2000 asset base.
- This demonstrates one of the feedback links in the ABC: The existence of the SS2000 product line and its architecture enabled new business opportunities.

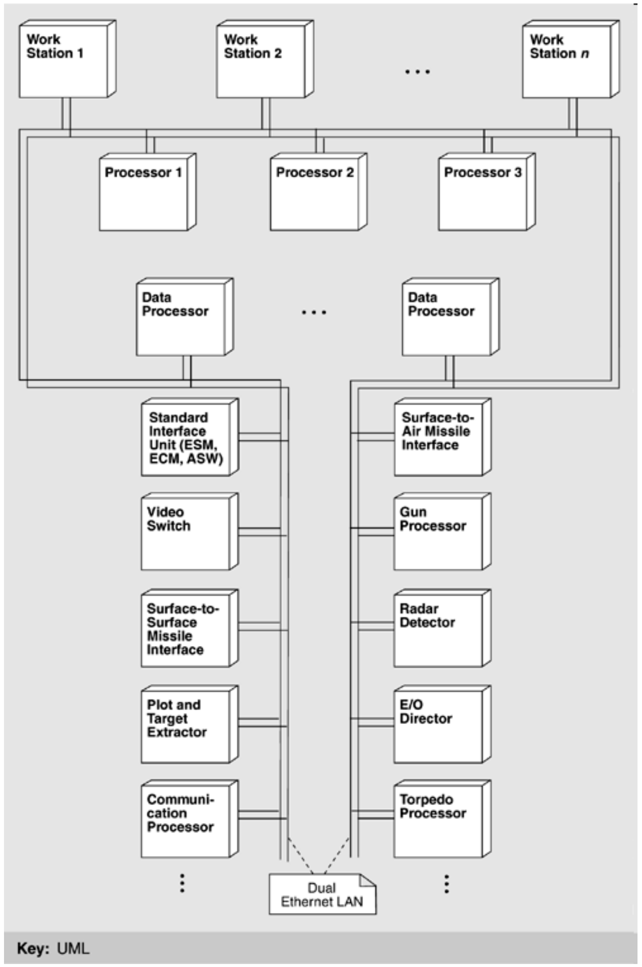
UNIVERSITY OF TECHNOLOGY SYDNEY

## THE PRODUCT LINE ARCHITECTURE

## Requirements and qualities

- As we have seen the primary purpose of an architecture is to achieve a system that meets its behavioural and quality requirements. The architecture for each SS2000 product line member was no exception. The most important of these requirements were:
  - Performance. Command-and-control systems must respond in real time to continuously arriving sensor inputs and be able to control weapons under tight deadlines.
  - Modifiability. The architecture needs to be robust with respect to computing platforms, operating systems, addition or replacement of sensor and weapon systems, human–computer interface requirements, communication protocols, and the like.
  - Safety, reliability, and availability. The system must be available when needed, provide the correct data and commands to weapon systems, and fire only under the correct conditions.
  - Testability. Each system must be integrable and testable so that errors (if any) are quickly found, isolated, and corrected.
- Besides these single-system requirements, the SS2000 architecture carried the additional burden of application to an entire class of systems. Thus its requirements included the ability to replace one module with another tailored to a particular system without disrupting the rest of the architecture.

# Physical architecture of the system



- Sensors and weapons deployed all over the ship
- Crew interact with sensors and weapons via distributed workstations
- Fault tolerant design
- Redundant LAN is the communications backbone
- A message bus architecture

## Architectural solution – Process view

- Each CPU runs a set of Ada programs;
- each Ada program runs on at most one processor.
- A program may consist of several Ada tasks.
- Systems in the SS2000 product line can consist of up to 300 Ada programs.

## The consequences of a distributed platform

- The requirement to run on a distributed computing platform has broad implications for the software architecture.
  - First, it necessitates building the system as a set of communicating processes, bringing the process view into play.
  - Having a process view at all means that the performance tactic "introduce concurrency" has been applied.
  - Distributed systems also raise issues of deadlock avoidance, communication protocols, fault tolerance in the case of a failed processor or communications link, network management and saturation avoidance, and performance concerns for coordination among tasks.
  - A number of conventions are used to support the distribution. These respond to the distributed requirements of the architecture as well as its product line aspects. The tasks and inter-component conventions include the following:

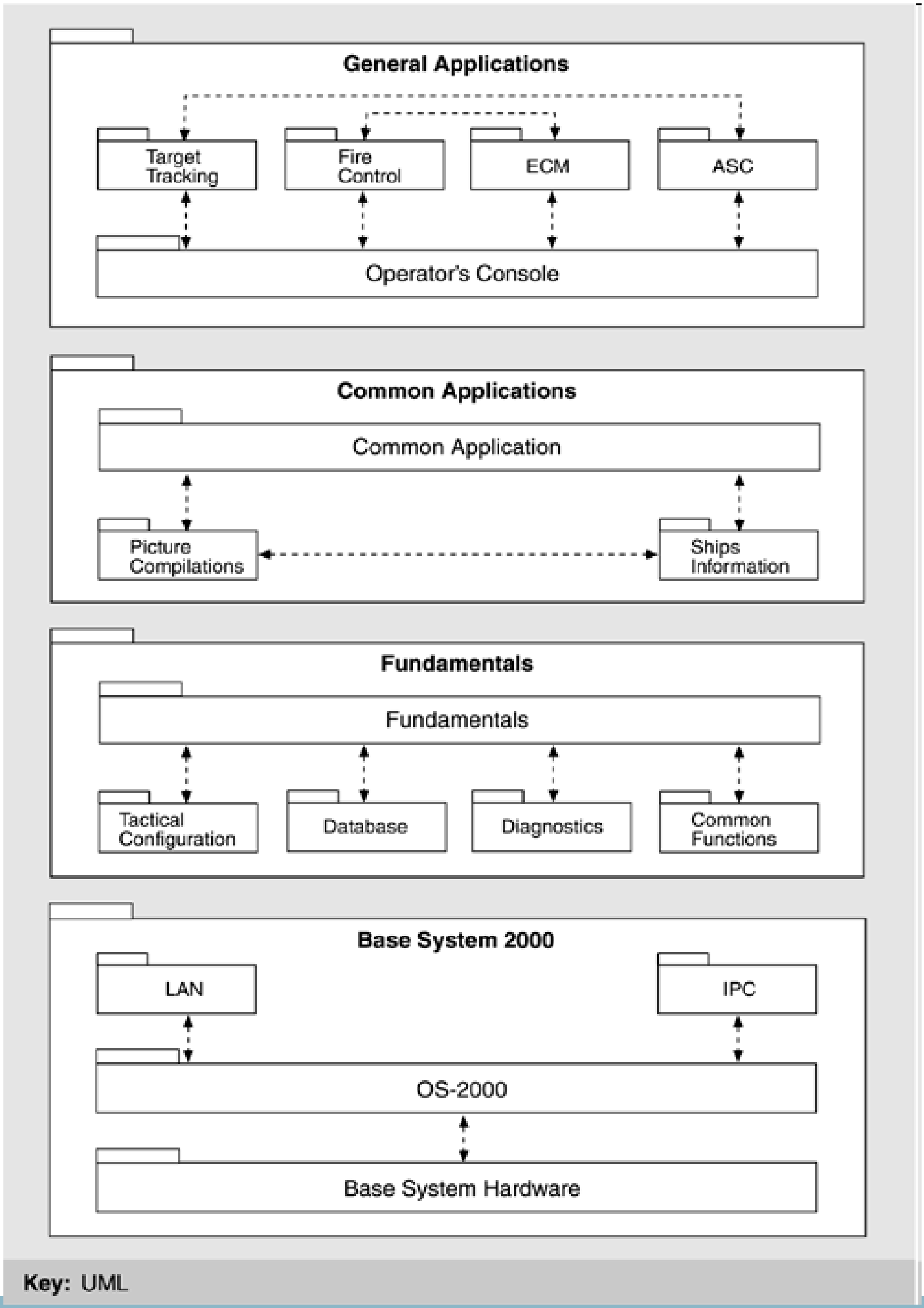
## Communication among processes

- Communication among components is by the passing of strongly typed messages. The abstract data type and the manipulating programs are provided by the component passing the message. Strong typing allows compile-time elimination of whole classes of errors. The message as the primary interface mechanism between components allows components to be written independently of each other's (changeable) implementation details with respect to data representation.
- Inter-process communication is the protocol for data transport between Ada applications that supports location independence, allowing communication between applications regardless of their residence on particular processors. This "anonymity of processor assignment" allows processes to be migrated across processors, for pre-runtime performance tuning and runtime reconfiguration as an approach to fault tolerance, with no accompanying change in source code.
- Ada task facilities are used to implement the threading model.

## The layered view

- The architecture for SS2000 is layered, as follows:
  - The grouping of modules is roughly based on the type of information they encapsulate.
    - Modules that must be modified if hardware platform, underlying LAN, or internode communication protocols are changed form one layer.
    - Modules that implement functionality common to all members of the family form another.
    - Modules specific to a particular customer product form a layer also.
  - The layers are ordered, with hardware-dependent layers at one end of the relation and application-specific layers at the other.
  - The layering is "strict," meaning that interactions among layers are restricted. A module residing in one layer can only access modules in its own or the next lower layer.

- S2000 layered architecture
  - The base layer provides an interface between applications and O/S, hardware and network
  - Base layer provides an interface to other applications without concern for the underlying platforms



## Summary

- Celcius Tech provides an example of successful product line architecture development
- The architecture and system have been extended beyond their original base of ships
- Recent reports are that Celcius Tech systems require have 3000 parameters