Software Architecture--Continued

- References for Software Architecture examples:
  - **Software Architecture, Perspectives on an Emerging Discipline**, by Mary Shaw and David Garlin, Prentice Hall, 1996.
  - B. Hayes-Roth, et al, *A Domain-Specific Software Architecture for Adaptive Intelligent Systems*, in **IEEE Trans. On Software Engineering**, vol. 21, no. 4, April 1995. pp 288-301

Another Software Architecture Example

- Mobile Robotics System
  - controls manned, partially-manned, or unmanned vehicle--car, submarine, space vehicle, etc.
  - System performs tasks that involve planning and dealing with obstacles and other external factors.
  - System has sensors and actuators and real-time performance constraints.

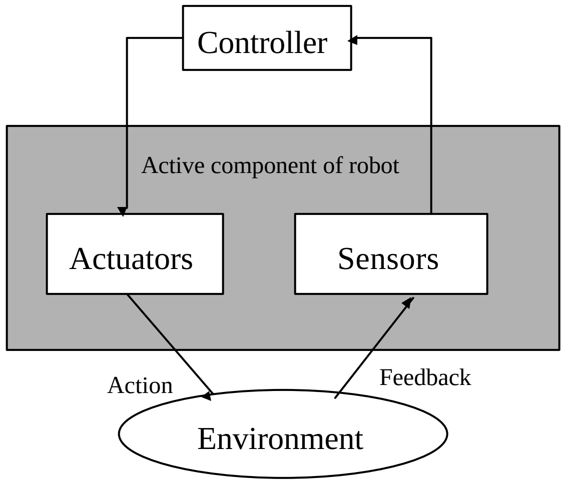
Mobile Robotics System--Requirements

- Req 1: System must provide both *deliberative* and *reactive* behavior.
- Req 2: System must deal with *uncertainty*.
- Req. 3 System must deal with *dangers* in robot's operation and environment.
- Req. 4: System must be *flexible* with respect to experimentation and reconfiguration of robot and modification of tasks.

Mobile Robots--Proposed Architectures

- Lozano--control loops
- Elfes--layered organization
- Simmons--task-control architecture
- Shafer-blackboard
- Hayes-Roth--AIS Reference Architecture
  - layers
  - pipes and filters

Mobile Robots--Control Loop Architecture



Control Loop Architecture--Strengths and Weaknesses

- Req 1--deliberative and reactive behavior
  - advantage-simplicity
  - drawback-dealing with unpredictability
    - feedback loops assumes continuous changes in environment and continuous reaction
    - robots are often confronted with disparate, discrete events that require very different modes of reactive behavior.
  - Drawback-architecture provides no leverage for decomposing complex tasks into cooperating components.

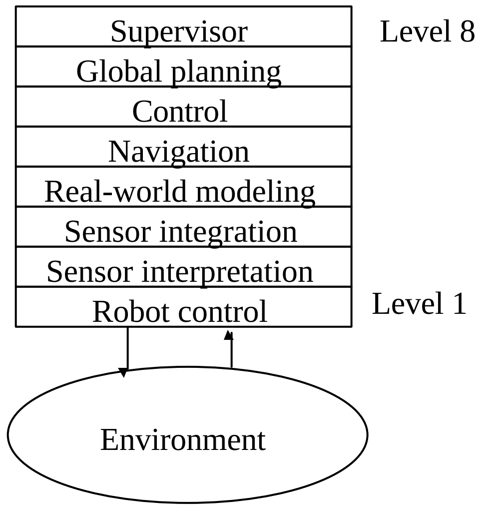
Control Loop Architecture--Continued

- Req 2--dealing with uncertainty
  - disadvantage-biased toward one way of dealing with uncertainty, namely iteration via closed-loop feedback.
- Req 3--safety and fault-tolerance
  - advantage-simplicity
  - advantage-easy to implement duplication (redundancy).
  - disadvantage-reaction to sudden, discrete events.

Control Loop Architecture--Continued

- Req 4--flexibility
  - drawback--architecture does not exhibit a modular component structure
- Overall Assessment: architecture may be appropriate for
  - simple systems
  - small number of external events
  - tasks that do not require complex decomposition,

Mobile Robots--Layered Architecture



Layered Architecture--Strengths and Weaknesses

- Req 1--deliberative and reactive behavior
  - advantage-architecture defines clear abstraction levels to guide desing
  - drawback-architectural structure does not reflect actual data and control-flow patterns
  - drawback-data hierarchy and control hierarchy are not separated.

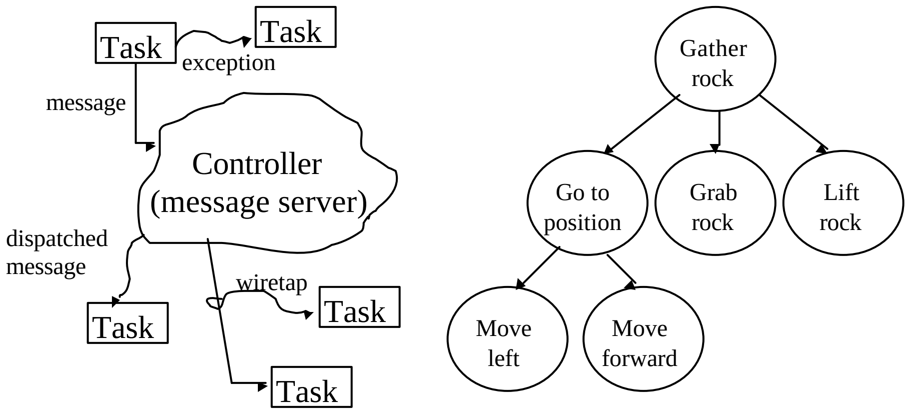
Layered Architecture--Continued

- Req 2--dealing with uncertainty
  - advantage-layers of abstraction should provide a good basis for resolving uncertainties.
- Req 3--safety and fault-tolerance
  - advantage-layers of abstraction should also help here.
  - drawback-emergency behavior may require short-circuiting of strict layering.

Layered Architecture--Continued

- Req 4--flexibility
  - drawback-changes to configuration and/or behavior may involve several or all layers
- Overall Assessment
  - layered model is useful for understanding and organizing system functionality
  - strict layered architecture may break down with respect to implementation and flexibility.

Mobile Robots--Task-control Architecture (Implicit Invocation)



Task-control Architecture--Three Functions

- Exceptions: override currently executing task in subtree that causes the exception
  - allow quick change of processing mode in response to spontaneous events
  - Exception handlers can abort or retry tasks
- Wiretapping: Tasks can eavesdrop on messages intended for other tasks.
- Monitors: Read information and execute action if data meets some criterion.

Task Control Architecture--Strengths and Weaknesses

- Req 1--deliberative and reactive behavior
  - advantage-task tress good for specifying deliberative behavior
  - advantage-exceptions/wiretapping/monitors good for reactive behavior
  - advantage-explicitly supports concurrency
  - drawback-- central controller may be a bottleneck.

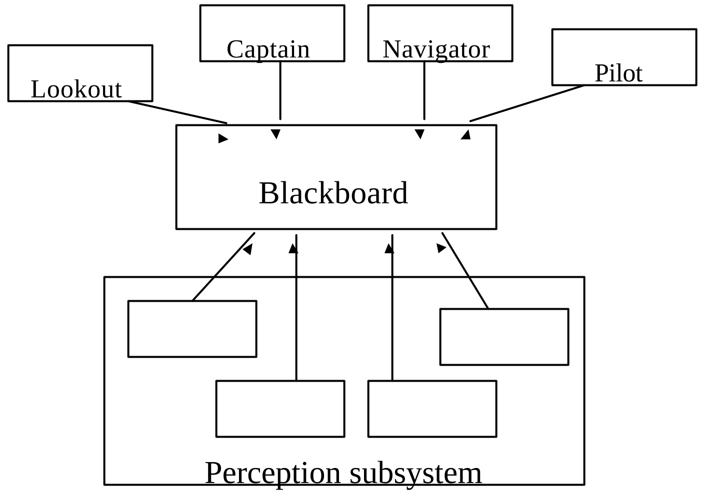
TCA Strengths and Weaknesses--Continued

- Req 2-- dealing with uncertainty
  - drawback: nothing in the architecture to explicitly address this. Task trees and exception handlers must accommodate all uncertainty
- Req 3--safety and fault-tolerance
  - advantage--exception handlers, wiretaps and monitors are provided.
  - advantage--easy to accommodate redundancy in task trees
  - disadvantage--controller may be weak link.

TCA Strengths and Weaknesses--Continued

- Req 4--flexibility
  - advantage--incremental development and replacement of components is straightforward
- Overall Assessment:
  - comprehensive set of features
  - appropriate for complex robot systems.

Mobile Robotics--Blackboard Architecture



Blackboard Architecture--Strengths and Weaknesses

- Req1-- Deliberative and reactive behavior
  - advantage: Easy to integrate disparate, autonomous subsystems
  - drawback: blackboard may be cumbersome in circumstances where direct interaction among components would be more natural.
- Req 2--Dealing with uncertainty
  - advantage: blackboard is well-suited for resolving conflicts and uncertainties.

Blackboard Strengths and Weaknesses--Continued

- Req3--safety and fault-tolerance
  - advantage: subsystems can monitor blackboard for potential trouble conditions (similar to wiretapping and monitoring in TCA architecture.
  - disadvantage: blackboard is critical resource.
- Req4--flexibility
  - advantage: blackboard is inherently flexible since subsystems retain autonomy.

Mobile Robotics--Summary of Architectural Tradeoffs

	Control Loop	Layers	TCA	Blackboard
Task coordination	+-	-	++	+
Dealing with uncertainty	-	+-	+-	+
Fault tolerance	+-	+-	++	+-
Safety	+-	+-	++	+
Performance	+-	-	++	+
Flexibility	-	-	+	+

Another Architecture Example

- Consider an application that needs to store objects to disk and restore them later.
  - This is the notion of a "persistent object".
- Application must be developed in an OO language that does not support persistence--e.g. C++
- Class structure will be subject to considerable modification over time.

Supporting Object Persistence--General Issues

- Storing or restoring an object requires specific knowledge of its internal state.
- When a class is modified, the mechanisms for storing and restoring objects of the class may also need to be modified.

Persistent Objects--Possible Architectural Approaches

- Approach 1: Application provides type-specific store and restore methods:

```
void storeA(classA:Obj);
void restoreA(classA:Obj);
  .
  .
  .
void storeZ(classZ:Obj);
void restoreZ(classZ:Obj);
```

Possible Architectural Approaches--Continued

- Approach 2: Provide a base class (or interface) for persistent objects, from which application classes are derived.
  - Derived classes override (or implement) store and restore methods.

Possible Architectural Approaches--Continued

- Approach 3: Provide a persistence mechanism that is independent of specific types.
  - This would have obvious advantages
  - But, is it feasible?
- Such an approach *may* be possible, but it requires the correct architectural approach.