TEAM API
FAQ

John Michaloski

Intelligent Systems Division
Manufacturing Engineering Laboratory
National Institute of Standards and Technology

TEAM PRESENTATION

- TEAM?
- TEAM Open-Architecture Background
  - terminology and definitions
  - OMAC module set
- TEAM API Specification Philosophy
  - Object Oriented, reusable assets, base and derived class strategy
- TEAM API Specification Language
  - C++, IDL, ISL, ROSE, ...
- How would you use the TEAM API?
- Discussion

What is an Application Programming Interface?

- What is an API?
  - API stands for Application Programming Interface, and refers to the programming front-end to a conceptual black box. When I say cos(x) I don't care how the cosine is implemented - table lookup or Taylor series. I just want a function "signature" and its calling and return sequence - assuming its behavior(implementation) is reasonable. For example, a cosine implemented using ASCII strings, may be very accurate, but take too long.

- Why is a (standard) API helpful?
  - programming complexity is reduced when there is only one alternative - instead of umpteen. For example, the cosine signature is generally accepted as cos(x), not cosine(x).

- What is TEAM API?
  - Group mission is to develop an API specification to **match the OMAC Open Architecture requirements** (enumerates standard controller components as well as life cycle and performance requirements).
  - Is there a homepage? http://isd.cme.nist.gov/info/teamapi

- What is TEAM?
  - Technology Enabling Agile Manufacturing is a multifaceted effort to improve manufacturing, headed by DOE with numerous industry, government and academic members. TEAM API is part of Intelligent Closed Loop Processing program.

What kind of OMAC requirements drive the TEAM API?

- use SERCOS drive instead of PWM/replace IO modules/replace control loop module; add tachometer feedback to the axes/add IO modules/control loop module;
- add secondary linear slide/add IO modules/add control loop module/replace axis group module; add spindle position control/add IO modules/add control module/replace axis group module;
- add tool changer/add IO modules/add logic control module; add error compensation/replace IO modules; add probing capability/add IO modules/add logic control module;
- add closed loop scanning probe/add IO modules/replace module; add acoustic emission sensor for tool breakage/add IO module/add logic control module; add real time image recognition of surface finish/add IO add logic control module;
- add LAN communications functions to support part program upload part program download, remote status monitoring and remote data acquisition;
- replace human interface face module with industry de facto standard operator interface; replace part program interpreter to support CL data generated from AutoCad or ProEngineer.
- NOTE - no mention of detailed functionality. Validation emphasis is on enabling plug and play, not did we forget the weave motion. Plug and play is being done, and we can do it too.

NIST

Why do we need a standard API?

- For example, at NIST, the Next Generation Inspection System (NGIS) project has integrated 3 commercial sensors and 1 generic sensor into the CMM controller.

- Taming diversity was a problem.: Each sensor had a different “front-end” or API - 1 had a .DLL driver interface, 1 had a memory mapped interface, 1 had a combination port and memory mapping. None of the sensors had the same API - that is, interface (i.e., data and control) format. This is not parsimonious.

- More applications could use sensors if the cost-economics of system integration are reduced. The latent market for sensors is probably double the current market. if development is simplified.and standardized (example ftp versus http). Now you can tap 3rd party use of sensors.

- Users are more likely to buy a sensor that conforms to some standard API+ to reduce integration, maintenance, and other life cycle costs. Integration and customization into an Open Architecture can be easier.

- Projects can save time and money reducing the proliferation of duplicated efforts. With sensors, you could use the same sensor interface code for two different projects and tailor each application based on sensor performance.

- Development of sensor tools (such as an analyzer) could result if the interfaces are standard across the industry.

What are the TEAM Modules?

Part Program Interpreter API	Axis API	Axis Group API
Task Coordination API	Human Interface API	Control Law API
Machine-to-Machine/ Shop Floor Communications API	IO Points API	User-Defined Process API

Sounds simple, why has it taken so long to specify the TEAM API?

- Defining a specification that is *cost-effective* and can achieve *real-time* performance is not trivial.
- Defining a *flexible* and *extensible* specification that can map into a broad range of platforms and implementations is problematic. A corresponding dilemma is unifying the mapping from specification to differing implementations.
- Black boxes (that is components) are not "self-contained." Defining an infrastructure within which the black boxes can operate is necessary and quite difficult.
- Open-Architecture concepts go beyond run-time APIs. There can be "other" APIs for configuration, integration, initialization, etc. Where the "other" API belong was the reason the TEAM API Workgroup spent time developing a Life Cycle.
  - For our cosine example, there is the need to specify whether the underlying math library is floating point or software emulation.
- Group dynamics can be a problem. Getting people to agree on anything can be a challenge.

What is the state of the TEAM Modules API definitions?

- Axis - agreed upon API
- Control Law - agreed upon API
- HMI - agreed upon API
- IO - agreed upon API, some question as to
- Machine-to-Machine - IDL sketched
- Part Program Translation - agreed upon basic common areas, some issues left
- Task Coordination - discussion closing in on common approach
- Trajectory Generator/Axis Group - holding pattern waiting for Task Coordination and Part Program Translation

Some TEAM API Terminology

- **Application Programming Interface** (API) - the protocol by which one interacts with a module.

- **Class** An abstract data type and inheritances.
  - For example, the Class SERCOS-Driven Axis describes objects in the running machine controller. A 3-axis mill would have three instantiations of that class; the 3 objects described by that class.

- **Object** - an instantiation of a class.
  - For Example: a probe object is an instantiated “sensor class” and a programmer interacts with the probe object using the “sensor class API.”

- **Component** -adopts the SOSAS concept of a reference architecture consisting of primitive and aggregate components. Components are defined as abstract building block elements that describe functionality and communication. The application architecture is built from these components. Components have the following attributes: responsibility; peer-to-peer or collaborative; relationships; and messages. (Say what?)
  - degrees of aggregation: cosine (& other primitives), math library, kinematics library, etc.

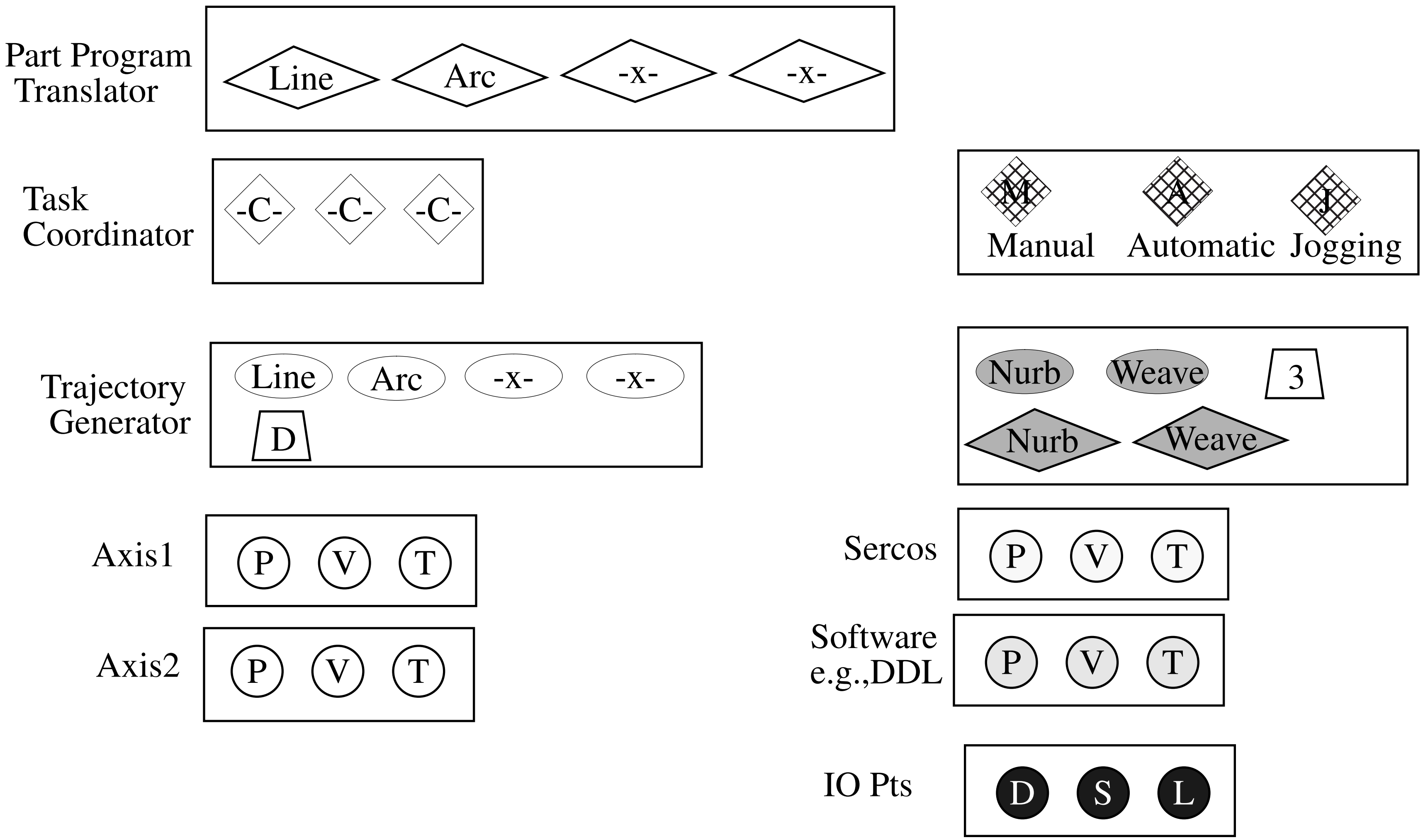
- **Module** - a piece of the system that is sufficiently defined such that it can be replaced by another piece, by a third party, with same service through the same interfaces, possible states, and conditions of state transitions.

What approach(es) were taken in defining the TEAM API?

- Adopt concepts of Object Oriented technology
  - class definitions containing both data and methods offer encapsulation (hide implementation from specification (API))
  - you can apply these notions quite adequately in non-OO languages, thank you very much
- Leverage OO notion of inheritance to use base and derived classes to add specialization (is-a relationship.)
  - When defining a control law, were do you stop? First PID, then Fuzzy, Neural Nets, Non-linear - need to contain realm of possibilities. The supertype is control law, each specialization is a subtype and each subtype inherits the definition of the supertype. At first, only define highly-demanded subtypes.
- Apply concept of foundation classes and aggregating components (has-a relationship.)
  - Analogy to foundation classes "building blocks" are sand, gravel, and clay while aggregating structural components would be bricks, walls, buildings. Use of these information models is required to define classes of increasingly broader scope.
  - Categories help organize the variety of information
- Foster the concept of reusable assets
  - self-contained or encapsulated, increases reliability, localizes changes, etc. - which reduces costs.
- Narrow focus of design interpretation using TEAM API Open Architecture Lifecycle

What level of integration is expected within TEAM API?

- TEAM API Open Architecture emphasis is low-level, for system integrator configuration - that is, linking .o's or connecting to OLE servers, not connecting programs.

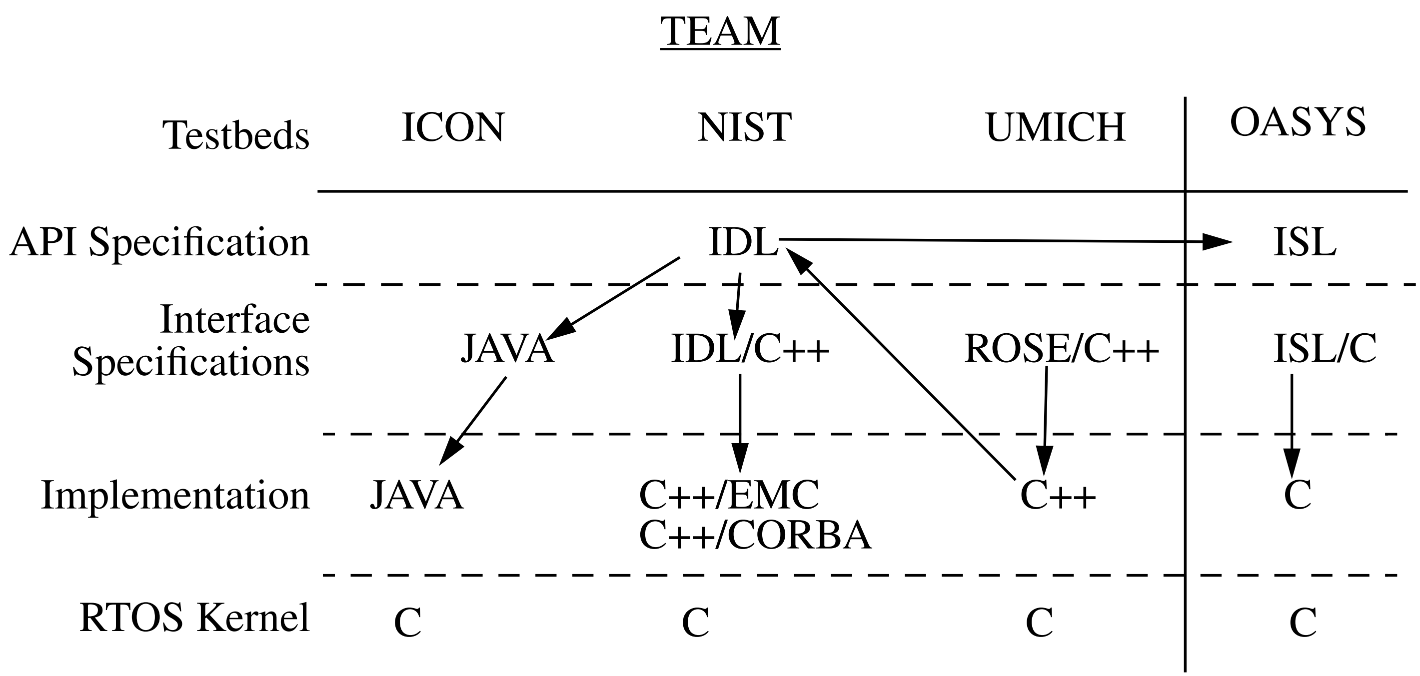


What language should I use to specify my API?

- Why can't I use English to specify the interfaces?
- C++ - can it be an interface specification and programming language too?
  - TEAM API plan to publish the C++ API (header files) - unclear if this is the standard.
- OMG Object Management Group
  - IDL: Interface Definition Language
  - CORBA: Common Object Request Broker (ORB) Architecture: (unify rpc)
- Xerox Parc Interface Language Unification Project (ftp://ftp.parc.xerox.com/pub/ILU)
  - ILU: Interface Language Unification
  - ISL: Interface Language Specification (supports IDL)
- ROSE - Commercial Product from Rationale, Inc.
  - Booche: object oriented modeling and specification
  - OMT - Rumbaugh Object Modeling Technique
  - generates C++
- JAVA

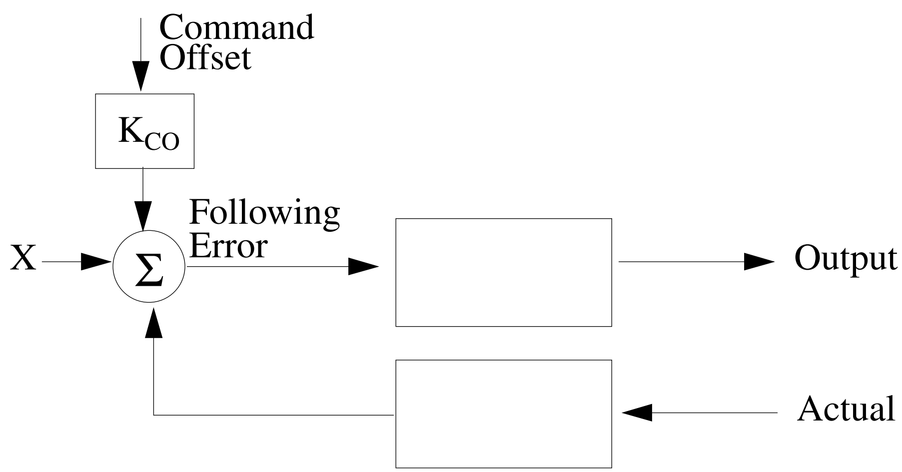
How do I get from Specification to Implementation?

- Problem unifying the mapping from Specification to Implementation
- TEAM API testbeds and OASYS - a related API effort



Where do I start when defining a base class?

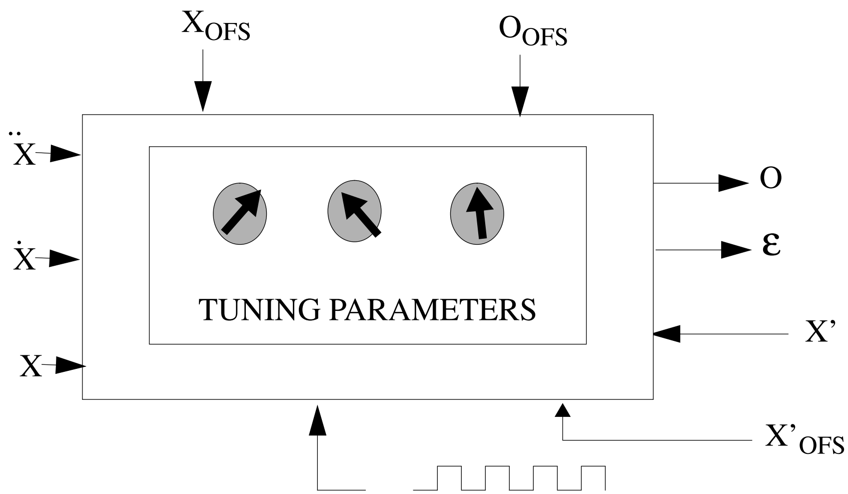
- Define the most basic functionality within each module base class
  - ***one step above "Input-Process-Output"***
  - for example, the basic definition of the control law module is sketched below



- Accommodating 3rd party signal injection must be explicitly addressed.
  - Pre-and post-offsets (or injection points) for which sensors or other process-related functionality can "tap" and dynamically modify behavior by applying some coordinate space transformation is provided for in all TEAM API modules.
  - The inclusion of a command offset is shown in the diagram to illustrate the ability to do sensor compensation.

What does a TEAM API Base Class look like?

- General Control Law Block Diagram - concept of tuning is encapsulated within black box and is conceptually controlled via “knob turning”

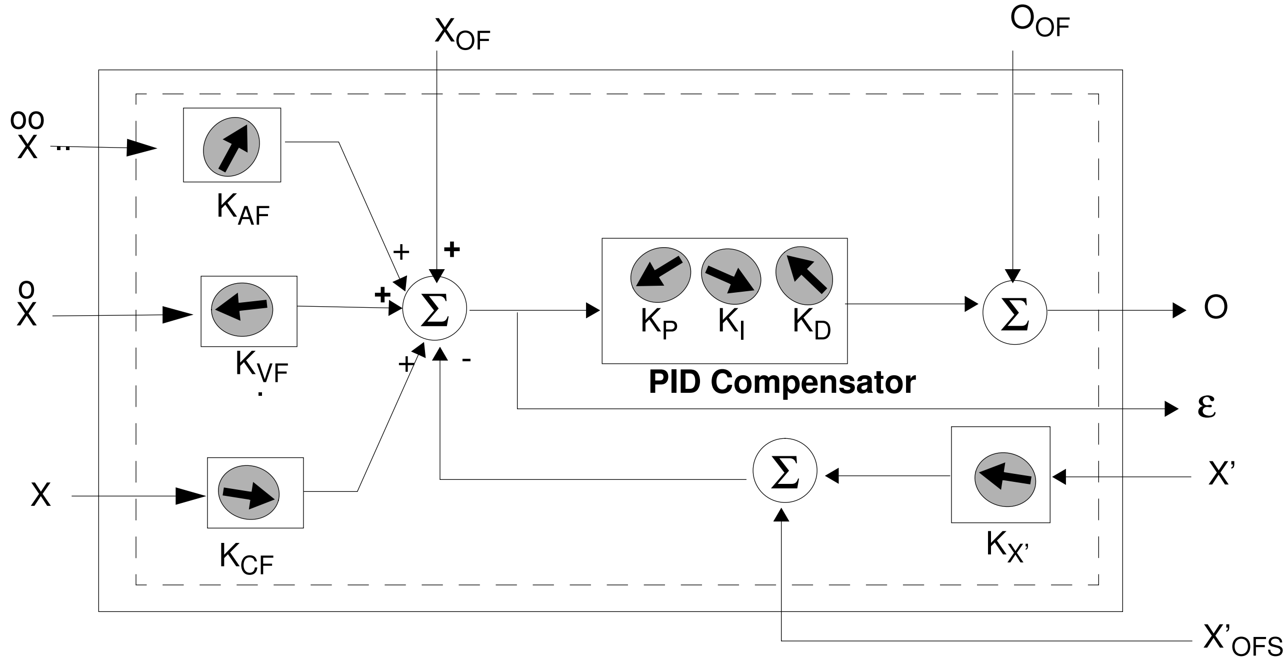


- IDL Control Law definition - assumes accessor functions to get and set object data members

```
interface CONTROL_LAW{
      // ATTRIBUTES
      attribute double x;
      attribute double xdot;
      attribute double xdotdot;
      attribute double output;
      attribute double actual;
      attribute double following_error;
      attribute double cmdOff, outOff,actOff, scalOff;};
      // OPERATIONS
      API::Status calculate_control_cmd();
      API::Status init();   // clear time history
};
```

How do I derive a specialized class?

- One adds functionality by deriving a specialized class - be it PID, Neural Net, Fuzzy Logic, etc.




```
// PID Example
interface PID_TUNING{
       // Attributes
       attribute double Kp,Ki,Kd;
       attribute double Kxdotdot,KxdotKx, Kxdotprime, Kactual, KactOff, KoutOff, KcmdOff;
}
// Example 1: Software Interface to PID Hardware Board
//interface PIDHard: PID_TUNING;
// Example 2: Software PID implementation
//interface PIDSoft: CONTROL_LAW, PID_TUNING;
// Example 3: naming P,V, T classes
class P : PID_CONTROL_LAW { };
```

How is aggregation used to get Reusable Data Assets?

Reusable Software Assets Library.
Example categories

Machining systems/cells, workstations | Plans

Simple machines, tool-changers, work changers ... | Processes

Axis groups | Fixtures. Other tooling

Machine tool axis or robotic joints
(translational; rotational)

Axis components *(sensors, actuators)* | Control components *(pid; filters)*

Geometry *(Coordinate frame; circle)* | Kinematic structure...

Units *(meter)* | Measures *(length)* | Containers *(matrix)* | ...

Primitive data types

How does one build an applications using TEAM API?

- The TEAM/ICLP API workgroup has defined standard API definitions using foundation classes. The goal of the foundation classes is to foster ``plug-and-play” component technology. The foundation classes form part of OA solution hierarchy:
  - Foundation Classes
  - Components
  - Integration Architecture
  - Application Architecture

- A good design analogy would be hardware component technology:
  - Foundation: hardware chips, RAM/ROM, CPUS, logic, etc.
  - Components: Memory banks are constructed from memory chips.
  - Integration Architecture: A hardware board (with data and memory addressing, control logic and ground wiring, as well as CPU/FPU/Clock, Memory, Control Logic, Bus Logic) is an integration architecture.
  - Application Architecture: computer board, memory board, device controller, etc.

- TEAM API workgroup does not specify the Integration Architecture which is critical to the applications.
  - potential candidates: agent-based, CORBA client/server, OLE, etc. - but would have to support hard real-time performance

TEAM API LifeCycle?

- Part 1 of the TEAM API work describes a lifecycle methodology needed for component development within an Open Architecture (OA) environment
  - what does it take to develop two OA implementations based on different design and implementation specifications using Commercial Off-the-Shelf (COTS)

- OA Life Cycle considers the roles from the following perspectives:
  - COMPONENT BUILDER must build their components to a standard specification and integration architecture (platform, os, compiler, infrastructure, etc.)
  - SYSTEM INTEGRATOR *selects* software components to match the application behavior and functional requirements, *configures* the components to match the application specification, *connects* the selected components using a integration architecture, *verifies* the system operation and *validates* compliance of components to the user-specification of performance and timing requirements
  - END USER application requirements *dictates* the selection of the tools and resources such as hardware, control devices, and computing platforms

What does application code look like?

- "SMALL PICTURE" - connect IO points between modules and then call module "execute function"
  - API developer defines code for each class as .DLL. Object naming & registry ignored.

```
// Most basic cyclic process - could be C++ template
closed_loop_process(){

// Read sensor - i.e, the current encoder value with IO system
io1.get_encoder(&value)

// Set the next actuator value
loop_closure1.set_actuator(value)

// Get the next value from the trajectory generator
trajgen1.get_axis1(&value)

// Use traj generator value to set next closed loop command
loop_closure1.set_command(value)
...

 // Use state to cause module to compute next value
if(state == running) loop_closure1.calculate_control_cmd();
 ...

// Read commanded output after it finishes cycle
 loop_closure.get_output(&value);

 // Put out value to DAC, (scaling done by io system)
 io2.put_DAC1(&value);
}

// initialize parameters
closed_loop.init(){
//1st time user binds to the objects io1, io2, loop_closure1, trajgen1
// name lookup is done in some global registry - either linker map or name server
```

What is the relationship between TEAM API and Standards?

- Standardization of TEAM API is under consideration.
- What about other API standardization efforts?
  - Collaboration with OSACA is underway.
  - Discussion with SEMATECH Computer Aided Integration Framework (CIMF) machine interface have been initiated.
  - No contact with OSEC or PLCOpen which is the industry group pushing IEC-1131, which is a suite of programming languages for implementations of discrete control, not interfaces to functionality.
- What about other controller standards?
  - IEC1131, RS274, BCL, etc. are application languages for part programming and logic sequencing. The TEAM API are application-independent but provide "hooks" for application-specific programming.
- What about computer industry infrastructure standards - CORBA, OLE, etc.?
  - TEAM API does not define an infrastructure. It is assumed that the selection of a platform, operating system, compiler and infrastructure could support the plug and play of TEAM API modules.

Where is TEAM API headed?

- Short-term goal is a draft standard specification
- Trying to fast-track and reach closure on as many modules as possible. More difficult ones will be slow-tracked.
- Trying to unify our effort with others
  - Tieing in with OSACA especially where TEAM API may be missing something
  - Tieing in with NEMI effort
- Trying to validate and test the accuracy, completeness, and flexibility of the specification
- Trying to tie up loose ends - IDL, ROSE, C++ unification