NASA Ames Research Center

Code TI

Planning and Scheduling Group

Autonomous Systems and Robotics Area

Interfacing External Systems

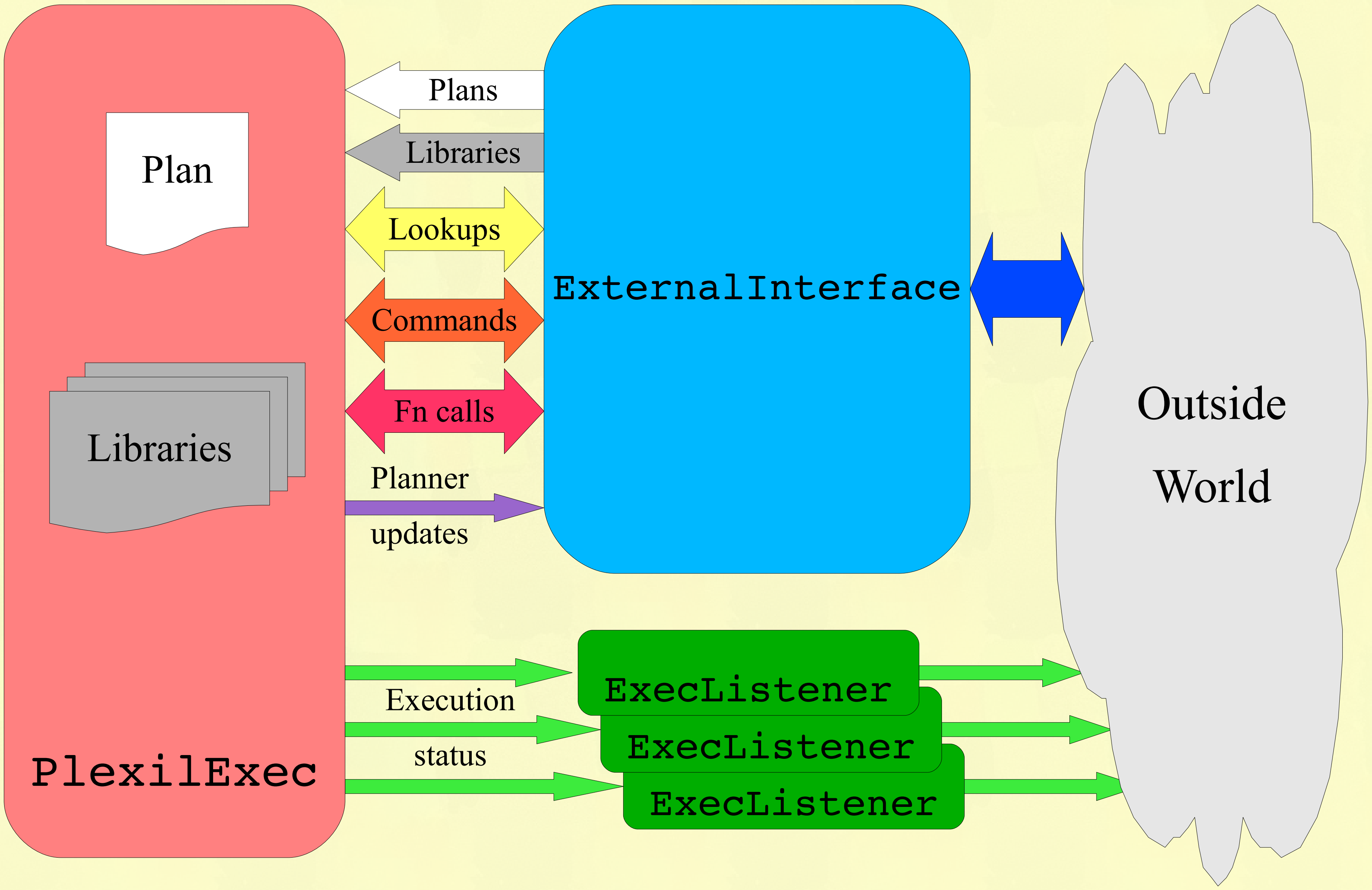
PLEXIL Workshop

July 2008

Interfacing External Systems

- Architectural overview
  - The `ExternalInterface` class
  - The `ExecListener` class
    - The `LuvListener` class
- Interfacing to the real world
  - The `ThreadedExternalInterface` class
  - The `AdaptorExecInterface` class
  - The `InterfaceAdaptor` class
  - Extending `InterfaceAdaptor`
  - Registering adaptors
  - Extending `ExecListener`
- Using CORBA
- Putting it all together
- Building an application

Architectural overview



The `ExternalInterface` class

- Virtual base class
- Defines API of interface to real world
- Singleton
- Implementation-agnostic (more or less)
  - Single or multiple threads
  - Monolithic or modular interface

The `ExecListener` class

- Abstract base class
- Defines API for reporting execution events
  - Node state transition
  - Plan added
  - Library node added
- Zero or more instances

The `LuvListener` class

- Concrete class derived from `ExecListener`
- Reports plan events to the Lightweight Universal-Exec Viewer (LUV)
- Communicates via TCP socket
- Can be used in any Universal Exec application

Interfacing to the real world

- Interface framework for real applications
  - The `ThreadedExternalInterface` class
  - The `AdaptorExecInterface` class
  - The `InterfaceAdaptor` class
- Delegates to `InterfaceAdaptor` instances
- Adaptors coded as application requires
- Adaptors can be added and removed dynamically

Class Diagram

The `ThreadedExternalInterface` class

- Concrete class derived from `ExternalInterface`
- Isolates the Exec from interfacing details
- Implements the `ExternalInterface` API
- Implements the `AdaptorExecInterface` API (see next slide)
- Multi-threaded
- Most input data is queued
- Delegates to `InterfaceAdaptor` instances
  - Adaptors indexed by:
    - Lookup (state) name
    - Command name
    - Function name
  - Delegates to default adaptor if none found by name

The `AdaptorExecInterface` class

- Abstract base class
- Implements Singleton design pattern
- Defines part of `ThreadedExternalInterface` API as seen by `InterfaceAdaptor`
- Isolates `InterfaceAdaptor` subclasses from needing to know implementation of `ThreadedExternalInterface`

The `InterfaceAdaptor` class

- Abstract base class
- Provides essential methods
- Has virtual methods for:
  - Lookups
  - Commands
  - Function calls
  - Planner updates
- Default methods print error message
- Subclasses implement these methods as needed by application
- Subclasses responsible for data format translation

Extending `InterfaceAdaptor`

- Partition functionality as desired
- Suggest one adaptor class per external device type
- Select communication method:
  - IPC
  - Socket
  - CORBA
  - ... etc.
- Adaptor instance is responsible for checking:
  - Name of operation(s)
  - Argument count and formats

Extending `InterfaceAdaptor`: Implementing lookup methods

- Lookups
  - `lookupNow()`
    - Should return immediately
    - Store results in 2nd argument (`std::vector<double> &`)
  - `registerChangeLookup()`
  - `registerFrequencyLookup()`
    - Set up asynchronous lookups (e.g. telemetry)
    - Values returned via `AdaptorExecInterface::handleValueChange()`
  - `unregisterChangeLookup()`
  - `unregisterFrequencyLookup()`
    - Perform cleanup when asynchronous lookups go out of scope

Extending `InterfaceAdaptor`: Implementing lookup methods, continued

- Bare minimum: implement `lookupNow()` for state `“time”`
  - Used internally by Exec
  - Can return `Expression::UNKNOWN()`
- Plans can use LookupOnChange of time to implement timers... conversely can be implemented *by* a timer
- If you implement `registerChangeLookup()` or `registerFrequencyLookup()`, you must also implement `lookupNow()` for same state(s)
  - Can simply return `Expression::UNKNOWN()`
- Call `AdaptorExecInterface::notifyOfExternalEvent()` after posting asynchronous lookup values

Extending `InterfaceAdaptor`: Commands, functions, planner update

- Commands
  - `executeCommand()`
  - `invokeAbort()`
- Functions
  - `executeFunctionCall()`
- Planner update
  - `sendPlannerUpdate()`
- Called in a batch after node transitions completed
- Acknowledgment can be delayed
- Post ack and return values to Exec with `AdapterExecInterface::handleValueChange()`
- Call `AdaptorExecInterface::notifyOfExternalEvent()` after posting acks and return values

Extending `InterfaceAdaptor`: Plans and libraries

- Use `AdaptorExecInterface` methods:
  - `handleAddPlan()`
  - `handleAddLibrary()`
- Translate from XML to intermediate representation (`PlexilNode`) with `PlexilXmlParser`
- Call `AdaptorExecInterface::notifyOfExternalEvent()` after new plan sent
  - Not needed for libraries

Extending `InterfaceAdaptor`: General hints

- Delayed return values (asynch lookups, commands, functions, etc.) and plans require call to `AdaptorExecInterface::notifyOfExternalEvent()`
- Exec does not run until this method is called
- Not needed by `lookupNow()` or libraries

Registering adaptors

- `AdaptorExecInterface` methods:
  - `setDefaultAdaptor()`
    - Use if only one adaptor needed
    - ... or if one adaptor handles most interfacing
    - Can use `DummyAdaptor` instance for debugging
  - `registerPlannerUpdateInterface()`
  - Others keyed by name:
    - `registerLookupInterface()`
    - `registerCommandInterface()`
    - `registerFunctionInterface()`

Extending `ExecListener`

- Methods
  - `notifyOfTransition()`
    - Called during quiescence cycle
    - One call per node state transition
    - Should be fast
  - `notifyOfAddPlan()`
  - `notifyOfAddLibrary()`
- Any or all of the above can be empty methods

Using CORBA

- Based on open source ACE/TAO implementation
- `CorbaHelper` class provides essential utilities:
  - `getOrb()`
  - `initializeOrb()`
  - `initializePOA()`
  - `initializeNameService()`
  - `queryNamingServiceForObject()`
  - `nameServiceBind()`
- `ExecListener` implementations
  - `EventChannelExecListener`
  - `NotificationChannelExecListener`
  - Extensible framework w/ user-definable filtering, formatting

Putting it all together

- Interface runs exec... not other way around!
- Exec is event-driven
- Nothing happens unless interface notifies Exec of a new event

Building an application

- Initialize Exec static data structures:
  - `initializeExpressions()`
  - `initializeStateManagers()`
- Construct `ThreadedExternalInterface` instance
- Construct at least one interface adaptor instance and register it with the external interface
- Construct the `PlexilExec` instance
- Attach the external interface to the exec
- Construct, register exec listener instances with exec as needed
- Start Exec by calling `ThreadedExternalInterface::run()` or `ThreadedExternalInterface::spawnExecThread()`

Questions?

- See the PLEXIL wiki: http://plexil.wiki.sourceforge.net/Interfacing
- See the PLEXIL doxygen web site: http://plexil.sourceforge.net/doxygen/universal-exec/
- “Use the source”
- Email: plexil-support@lists.sourceforge.net