

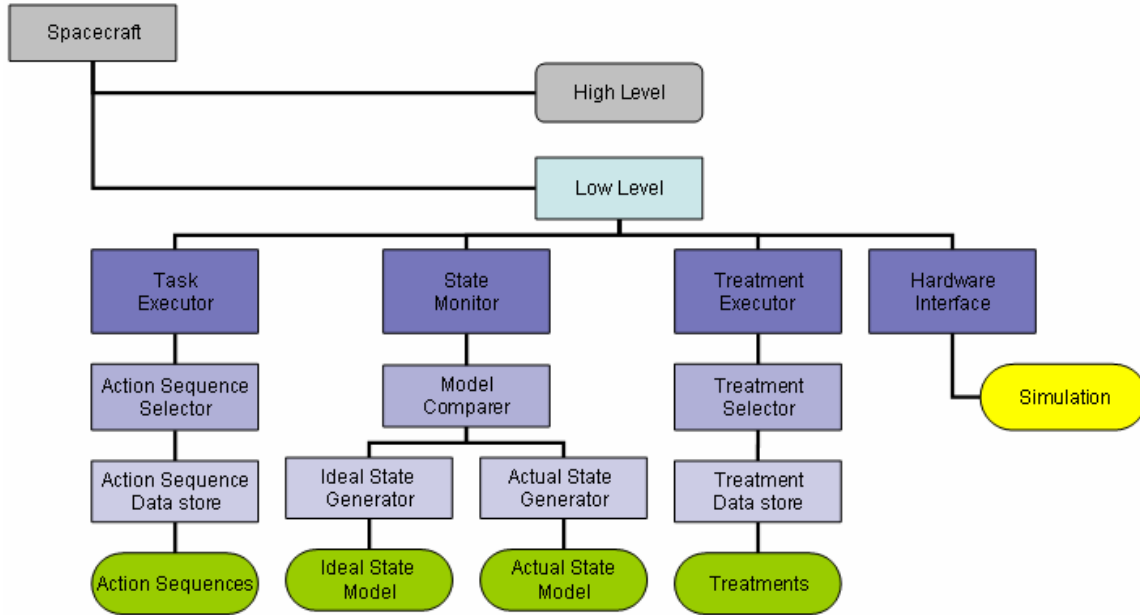
1. **Code**
2. **Required Component Methods and Component Hierarchy**
3. **Main Component Communication**
4. **Config File for Spacecraft**
5. **Replacing a System Component**
6. **Action Sequences**
7. **Logging**

Code

1. The root directory for the source code is *src*
2. Three directories inside the *src* directory which are the package directories
 - 2.1. *highlowinterface*
 - 2.1.1. contains classes used for communicating between the high-level and low-level systems.
 - 2.1.2. package name gov.nasa.gsfc.ants.highlowinterface
 - 2.2. *framework*
 - 2.2.1. contains all the framework classes (i.e. all abstract classes and interfaces)
 - 2.2.2. contains utility classes such as the logging class
 - 2.2.3. package name gov.nasa.gsfc.ANTS2.framework
 - 2.3. *components*
 - 2.3.1. contains implementation of framework
 - 2.3.2. package name gov.nasa.gsfc.ANTS2.components

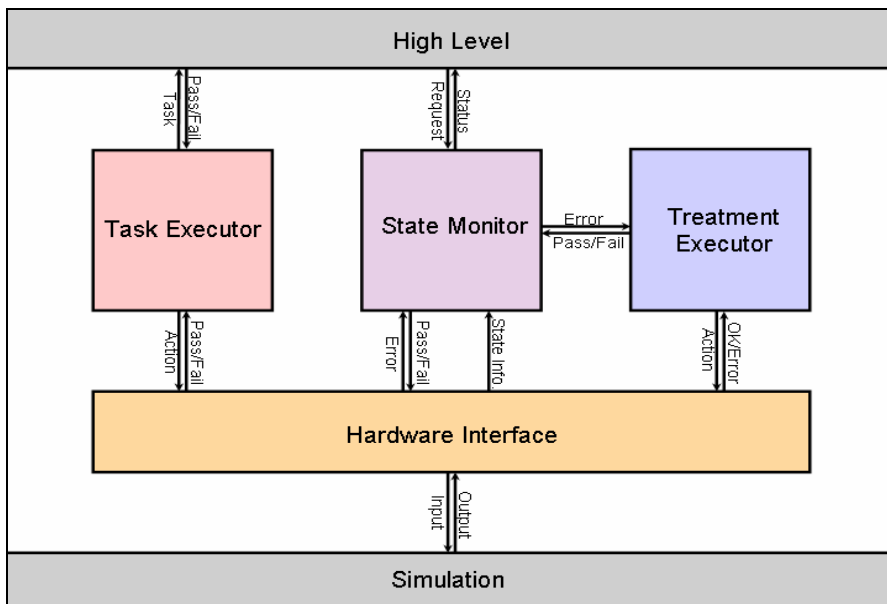
Required Component methods and Component Hierarchy

Framework Class	Required Methods
AbstractLowLevel	public boolean executeTask(Task t); public boolean executeTask(Task t); public void shutdown();
AbstractTaskExecutor	public boolean executeTask(Task t); public void suspend(); public void resume(); public void abortTask(); public void setActionExecutorInterface(ActionExecutorInterface ae);
AbstractStateMonitor	public State getState();
AbstractTreatmentExecutor	public boolean executeTreatment(Feedback feedback); public boolean executeTreatment(Feedback feedback, StateModel actualState); public TreatmentHandlerInterface getTreatmentHandlerInterface();
AbstractHardwareInterface	protected Feedback sendCommand(Action action); public void closeConnection(); public boolean executeAction(Action action); public ActionExecutorInterface getActionExecutorInterface(); public Feedback executeActionFeedback(Action action); public ActionExecutorFeedbackInterface getActionExecutorFeedbackInterface();
AbstractTasktoActionSequenceMapper	public ActionSequenceInterface getActionSequence(Task task);
AbstractStateComparer	public State getActualState(); public StateModel getActualStateModel(); public boolean compareStates();
AbstractErrorToTreatmentMapper	public boolean executeTreatment(Feedback feedback); public boolean executeTreatment(Feedback feedback, StateModel actualState); public TreatmentHandlerInterface getTreatmentHandlerInterface();
ActionSequenceDataStoreInterface	public ActionSequenceInterface[] getActionSequences(Task task);
TreatmentDataStoreInterface	public TreatmentInterface[] getTreatments(Feedback feedback); public TreatmentInterface[] getTreatments(Feedback feedback, StateModel state); public TreatmentInterface[] getTreatments(StateModel state);
AbstractStateGenerator	public StateModel getStateModel(); public State getState();



Main Component Communication

1. Done through Interfaces (Sender/Receiver/Controller architecture)
 - 1.1. Sender – sends messages
 - 1.2. Receiver – receives messages
 - 1.3. Controller – sets the sender receiver relationship



Example: Task Executor sends action to the Hardware Interface

Task Executor is the sender, must implement the ActionIssuerInterface:

```

public class TaskExecutorComponent extends AbstractTaskExecutor implements
ActionIssuerInterface ...
{
    ...
    public void setActionExecutorInterface(ActionExecutorInterface aei)
    { ... }
    ...
}

```

HardwareInterface is the receiver, must implement the ActionExecutorInterface:

```

public class HardwareInterfaceComponent extends AbstractHardwareInterface
implements ActionExecutorInterface ...
{
    ...
    public boolean executeAction(Action action)
    { ... }

    public ActionExecutorInterface getActionExecutorInterface()
    { ... }
    ...
}

```

Since the Task Executor and the Hardware Interface are part of the low level the Low Level class is the controller and is responsible for connecting the two.

```

public class LowLevel extends AbstractLowLevel implements
ActionIssuerExecutorController ...
{
    ...
    public LowLevel()
    {
        taskExecutor = new TaskExecutorComponent();
        hardwareInterface = new HardwareInterfaceComponent();
        ...
        setActionIssuerExecutorInterface(taskExecutor, hardwareInterface);
        ...
    }

    public void setActionIssuerExecutorInterface(
        ActionIssuerInterface actionIssuerInterface,
        ActionExecutorInterface actionExecutorInterface)
    {
        actionIssuerInterface.setActionExecutorInterface(actionExecutorInterface);
    }
}

```

Config File for Spacecraft

1. Written in XML
 - 1.1. allows for comments
 - 1.2. self validating
2. Root element is the <Spacecraft> tag
3. High-level information is placed between the <HighLevel> tags

4. Low-level information is placed between the <LowLevel> tags

```
<Spacecraft>
  <HighLevel>
    ...
  </HighLevel>
  ...
  <LowLevel>
    <ClassName params="Class construct params">
      Low Level System Class Name
    </ClassName>

    <Components>
      ...
      <Component>
        <ClassName params="Class constructor params">
          Component Class Name
        </ClassName>
      </Component>
      ...
    </Components>
  </LowLevel>
</Spacecraft>
```

Replacing a Component Example: replacing the Task Executor component

1. Create a new class that extends the AbstractTaskExecutor class
2. Implement the appropriate interfaces
3. Change class name in the config file.

Initial config file:

```
<Spacecraft>
  ...
  <LowLevel>
    <ClassName>LowLevel</ClassName>
    <Components>
      ...
      <Component>
        <ClassName> TaskExecutorComponent </ClassName>
      </Component>
      ...
    </Components>
  </LowLevel>
</Spacecraft>
```

Create a new class that extends the AbstractTaskExecutor and compile it:

```
public class TaskExecutor2 extends AbstractTaskExecutor implements ...
{
  public boolean executeTask(Task t)
  {...}
  public void suspend()
  {...}
  public void resume()
```

```

{...}
public void abortTask()
{...}
public void setActionExecutorInterface(
    ActionExecutorInterface aei)
{...}
}

```

Change the class name in the config file:

```

<Spacecraft>
...
<LowLevel>
  <ClassName>LowLevel</ClassName>
  <Components>
    ...
    <Component>
      <ClassName> TaskExecutor2 </ClassName>
    </Component>
    ...
  </Components>
</LowLevel>
</Spacecraft>

```

Action Sequences

1. Action sequences are represented by classes
2. An action sequence is selected based on its runability and efficiency
 - 2.1. runability is determined by looking at the current state
 - 2.2. efficiency is calculated by the number of success divided by the number of runs
3. First check to see if an action sequence can be run given the current state of the spacecraft
4. Next select the action sequence with the best efficiency rating
 - 4.1. An action sequence is randomly selected if there is more that one with the same efficiency level.
5. Task to action sequence mapping are stored in a config file

Config file example:

```

:MapSky:
gov.nasa.gsfc.ANTS2.components.MapSky,34
gov.nasa.gsfc.ANTS2.components.MapSky2,25
:SetTrajectory:
gov.nasa.gsfc.ANTS2.components.SetTrajectory,50

```

Logging

1. Different log levels
 - 1.1. 0 – nothing gets logged
 - 1.2. 1 – just the low-level system
 - 1.3. 2 – four main components
 - 1.4. and so on
2. Log example attached