US 20130283163A1

(19) United States

(12) Patent Application Publication

Mehringer et al.

(10) Pub. No.: US 2013/0283163 A1

(43) Pub. Date: Oct. 24, 2013

(54) **INTUITIVE INTERFACE INITIATIVE**

(71) Applicants: **Todd Mehringer**, Jasper, IN (US); **Bret Mehringer**, Jasper, IN (US); **Peter Avenatti**, Bloomington, IN (US); **Mark Rondina**, Bloomfield, IN (US)

(72) Inventors: **Todd Mehringer**, Jasper, IN (US); **Bret Mehringer**, Jasper, IN (US); **Peter Avenatti**, Bloomington, IN (US); **Mark Rondina**, Bloomfield, IN (US)

(21) Appl. No.: **13/863,051**

(22) Filed: **Apr. 15, 2013**

Related U.S. Application Data

(60) Provisional application No. 61/637,499, filed on Apr. 24, 2012.

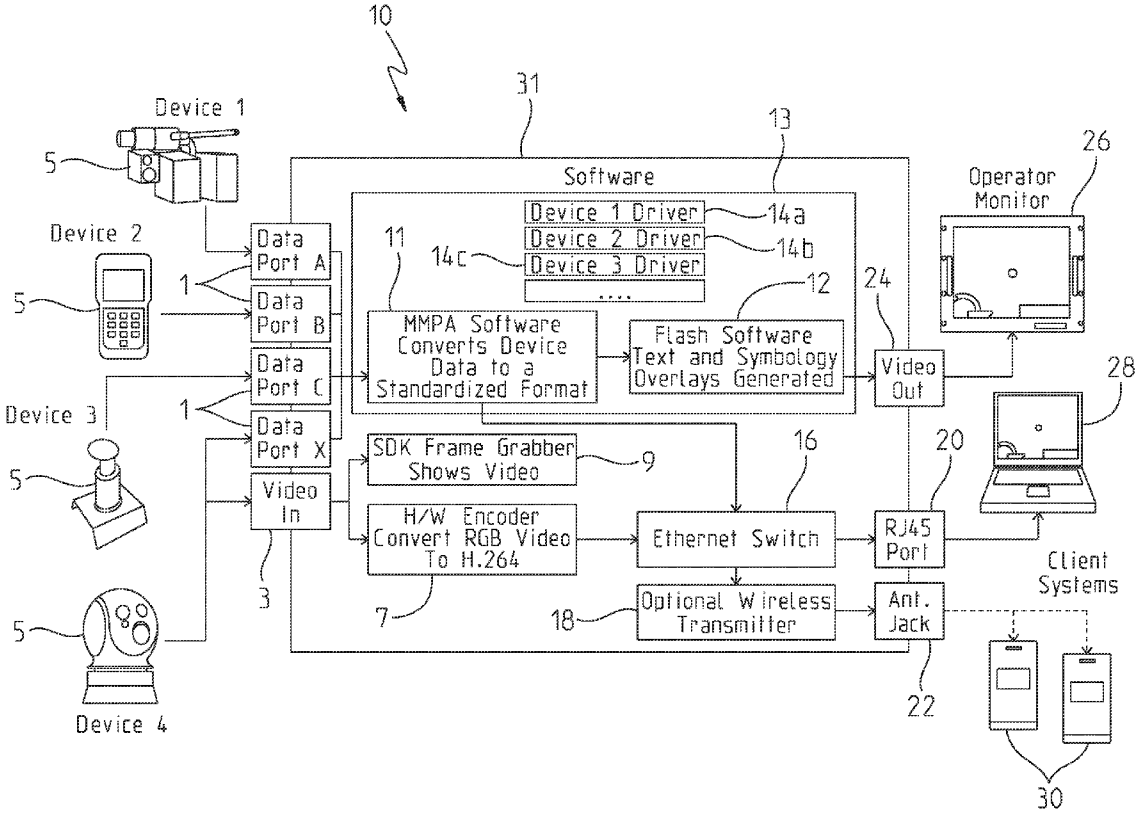
Publication Classification

(51) **Int. Cl.**
G06F 3/048 (2006.01)

(52) **U.S. Cl.**
CPC *G06F 3/048* (2013.01)
USPC **715/719**

(57) **ABSTRACT**

A device or computer program that takes the data and/or video from separate sensory devices and then combines this information for display in a standardized, but user configurable Graphical User Interface. Since each input device is paired with a device driver to properly parse the data for the input device, new devices may be added by writing new device drivers. The GUI has a standard set of elements that the individual user can modify to suit their personal preferences and needs. Since the Graphical User Interface is standardized, the user knows where to look for the data that they need no matter which sensory device is attached.



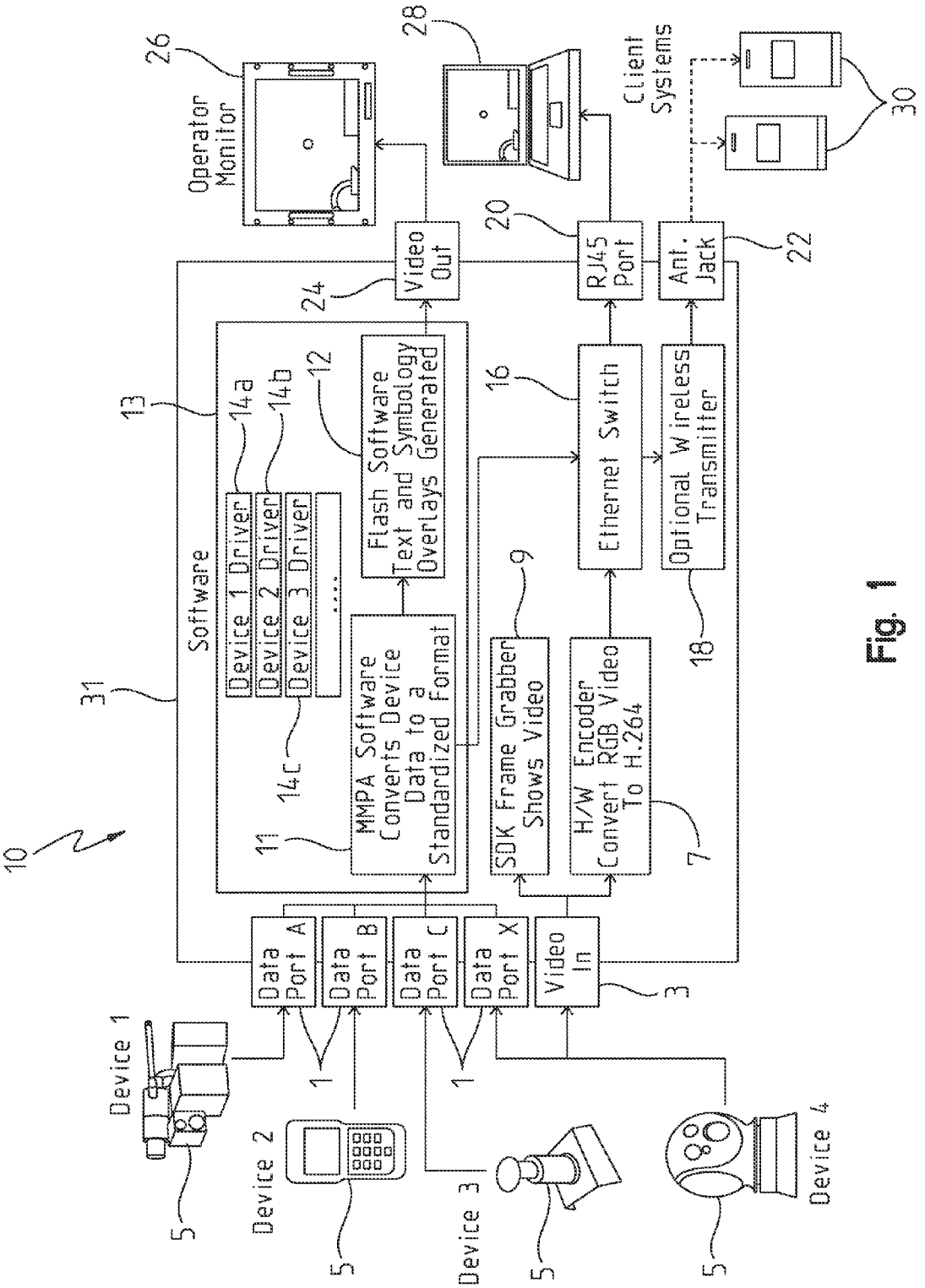

10
31
Software
13
14a
14b
14c
Device 1 Driver
Device 2 Driver
Device 3 Driver
....
12
Flash Software Text and Symbology Overlays Generated
11
MMPA Software Converts Device Data to a Standardized Format
9
SDK Frame Grabber Shows Video
7
H/W Encoder Convert RGB Video To H.264
16
Ethernet Switch
18
Optional Wireless Transmitter
24
Video Out
20
RJ45 Port
22
Ant. Jack
26
Operator Monitor
28
Client Systems
30
Data Port A
Data Port B
Data Port C
Data Port X
Video In
3
1
1
Device 1
5
Device 2
5
Device 3
5
Device 4
5

Fig. 1

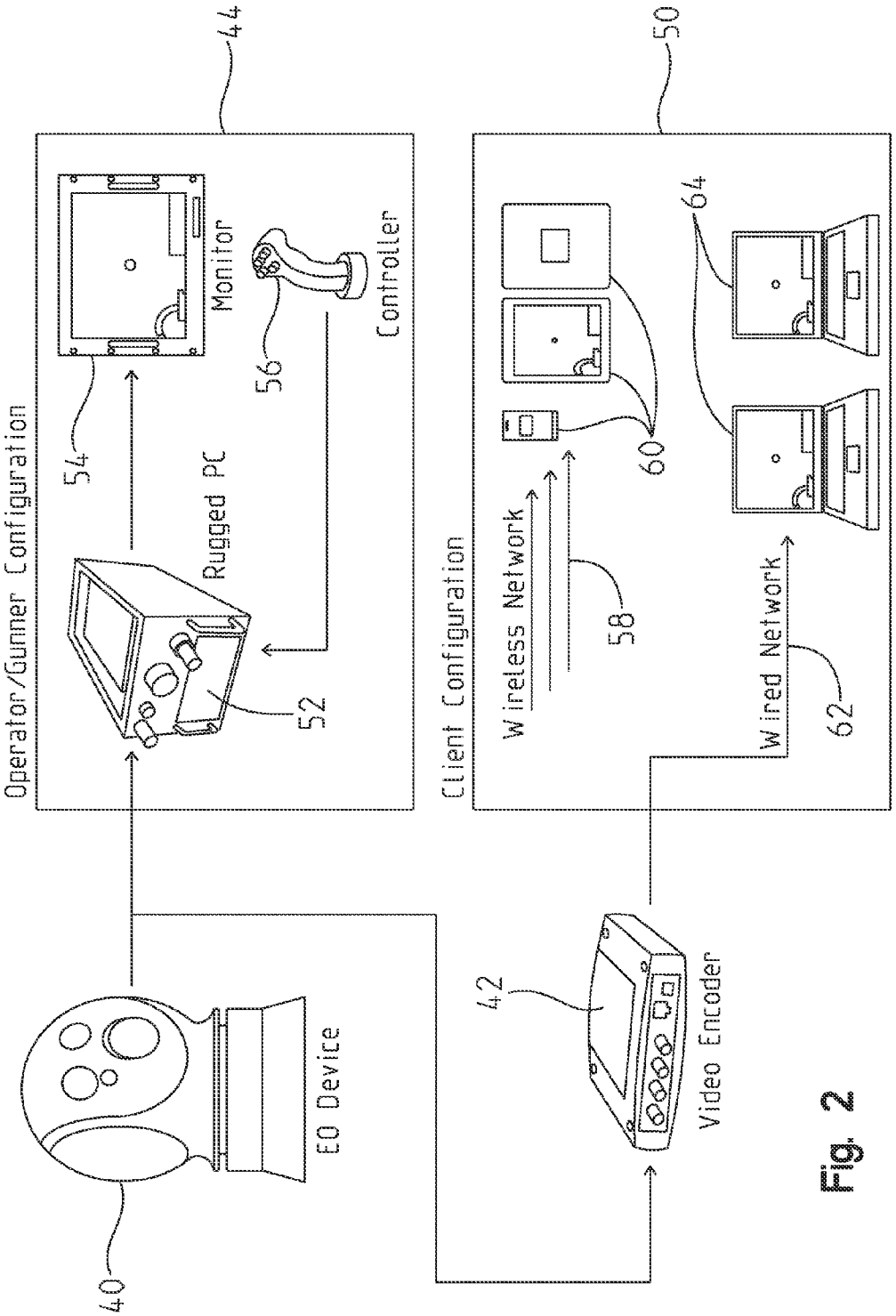

Operator/Gunner Configuration
44
Monitor
54
Controller
56
Rugged PC
52
Client Configuration
50
Wireless Network
58
60
64
Wired Network
62
EO Device
40
Video Encoder
42

Fig. 2

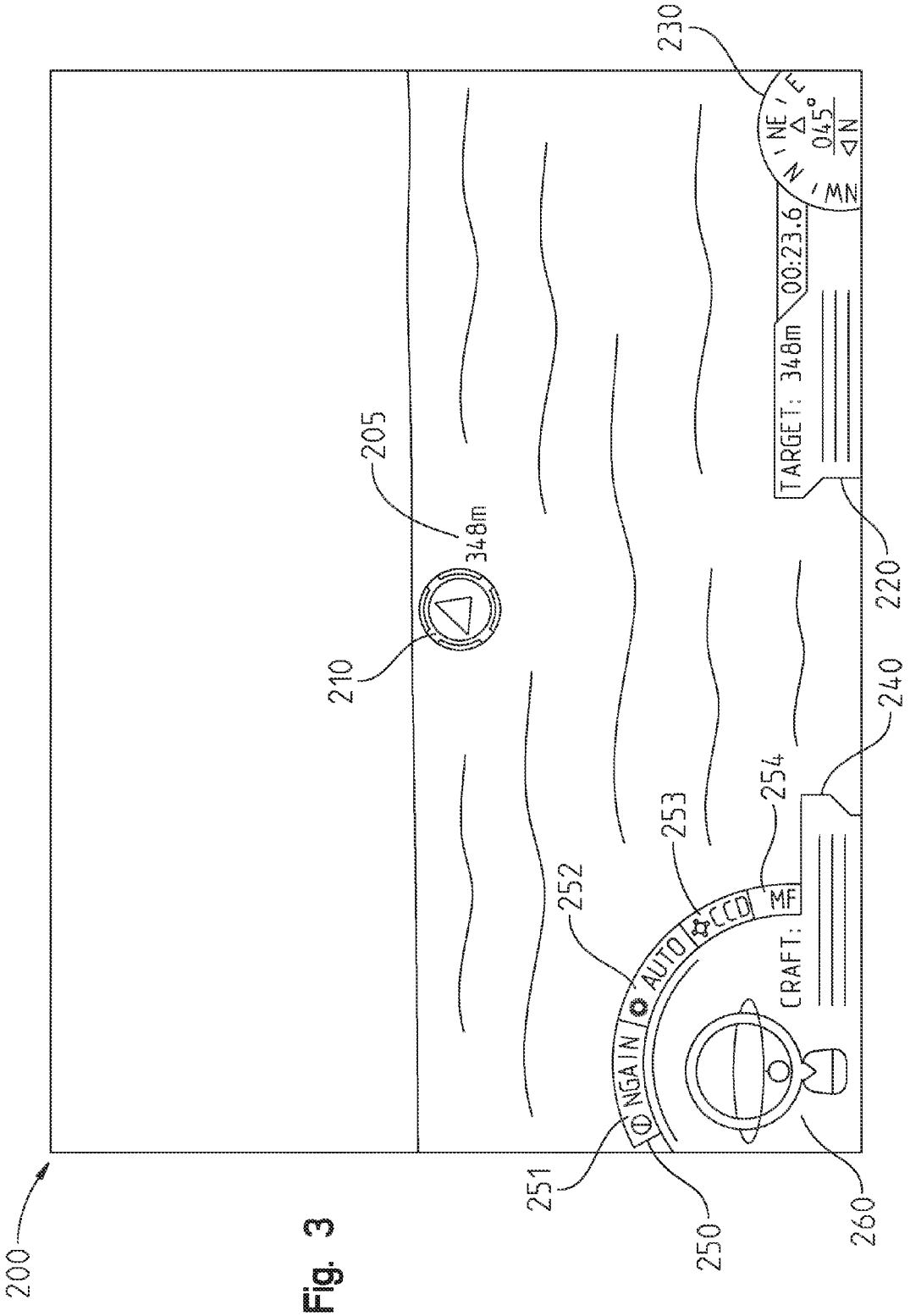

200
210
205
348m
230
N
NE
E
NW
045°
N
TARGET: 348m
00:23.6
220
240
254
253
252
MF
CCD
AUTO
NGAIN
CRAFT:
251
250
260

Fig. 3

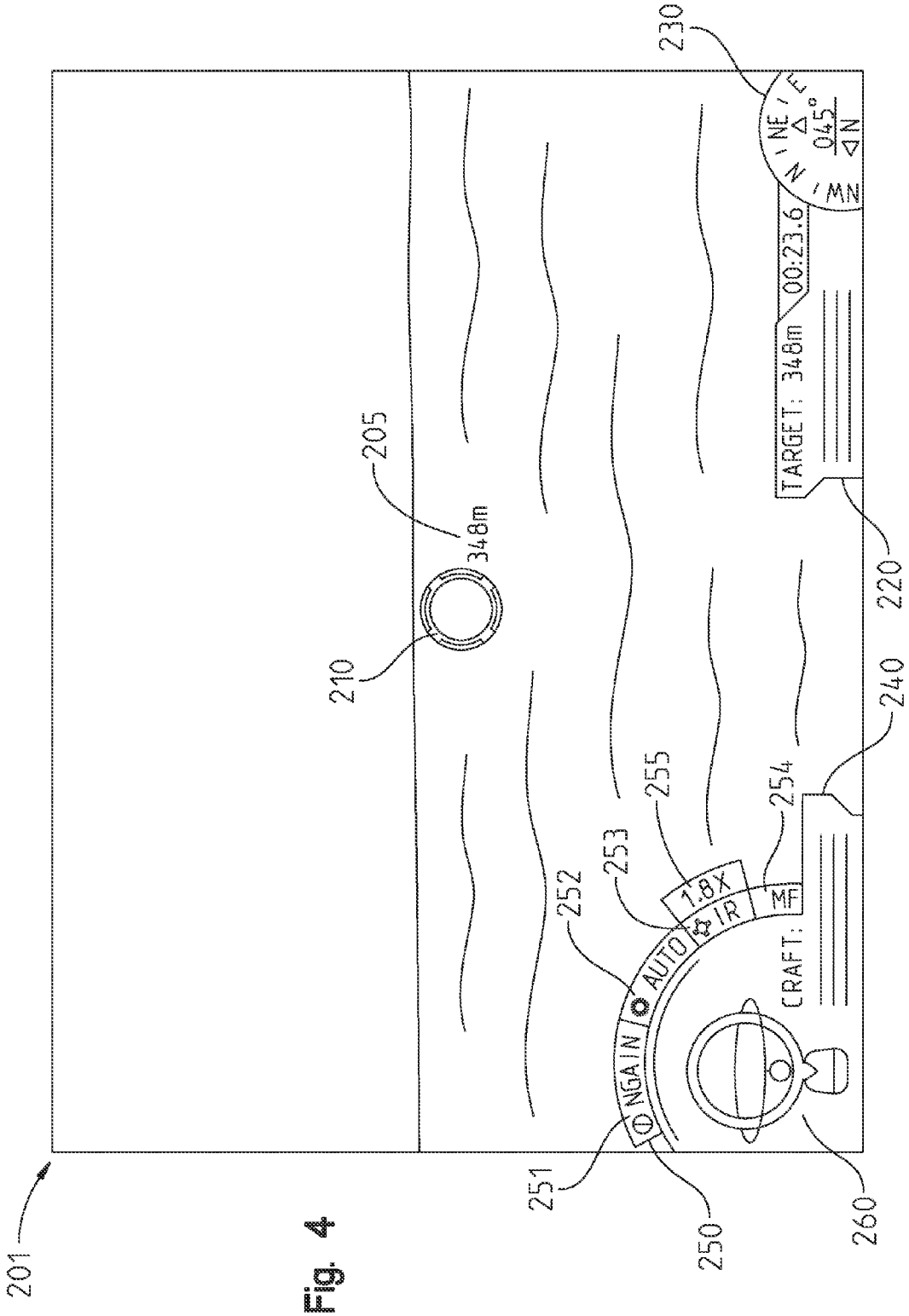

201
205
348m
210
230
NE
E
N
NW
045°
N
TARGET: 348m
00:23.6
220
240
CRAFT:
MF
IR
1.8X
AUTO
NGAIN
251
252
253
254
255
250
260

Fig. 4

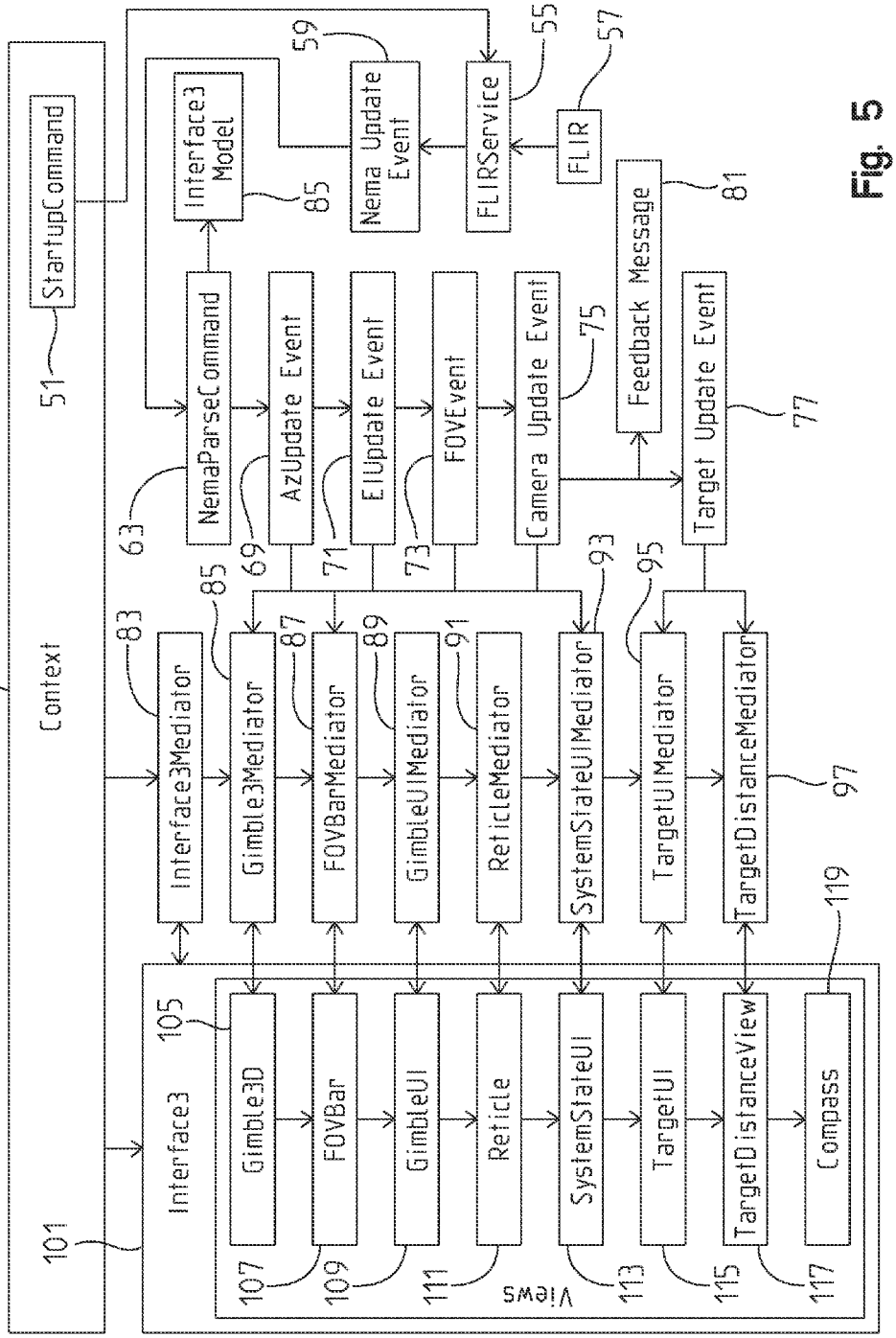


Fig. 5

3I Programming Architecture

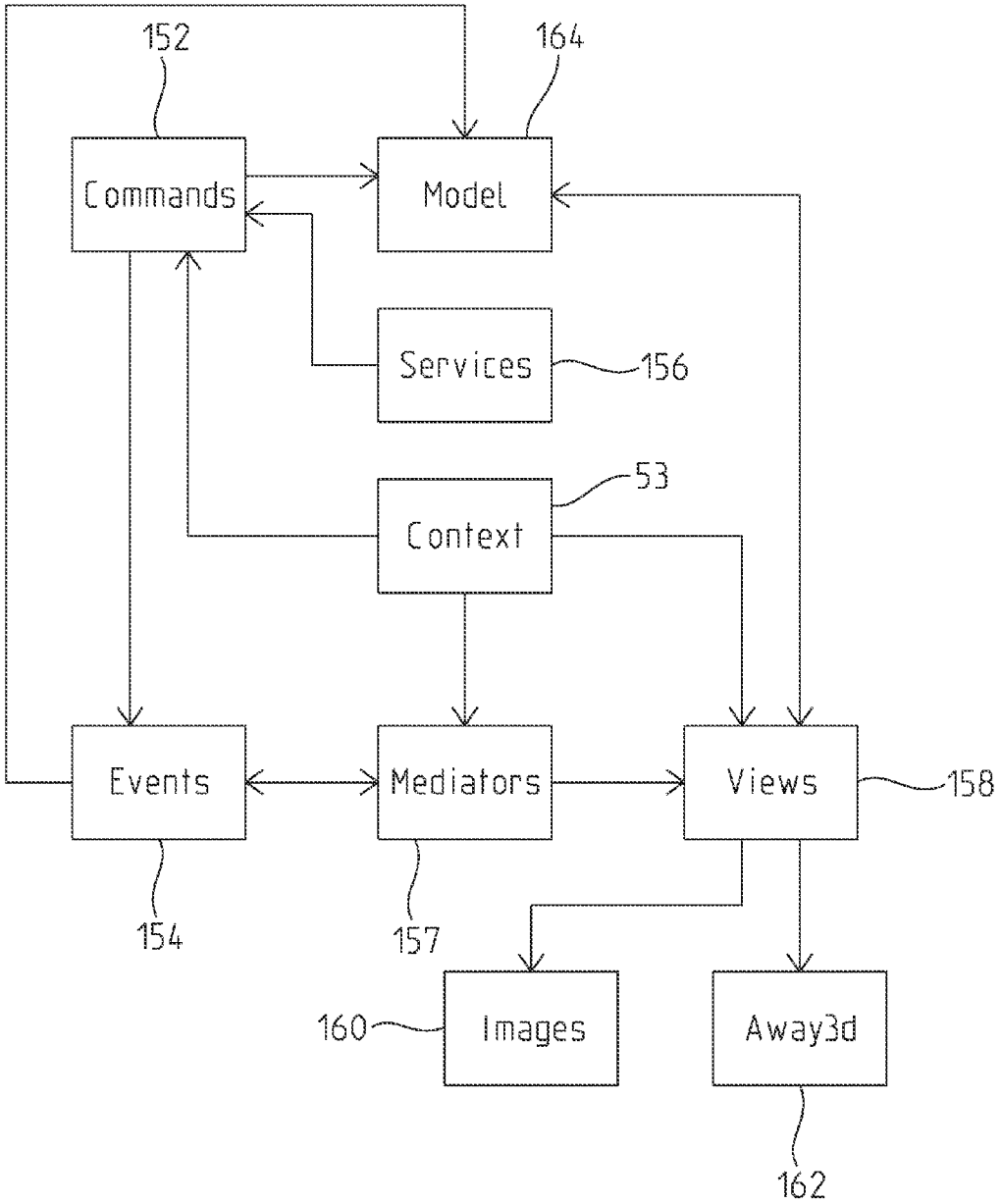

152
164
Commands
Model
Services
156
53
Context
Events
Mediators
Views
158
154
157
160
Images
Away3d
162

Fig. 6

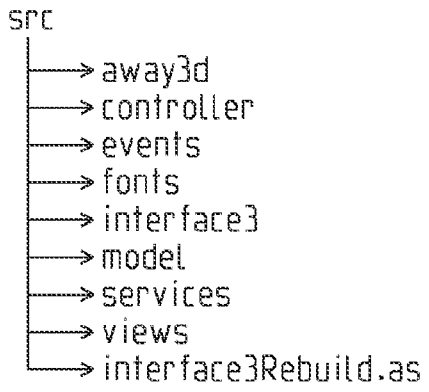

src
away3d
controller
events
fonts
interface3
model
services
views
interface3Rebuild.as

Fig. 7

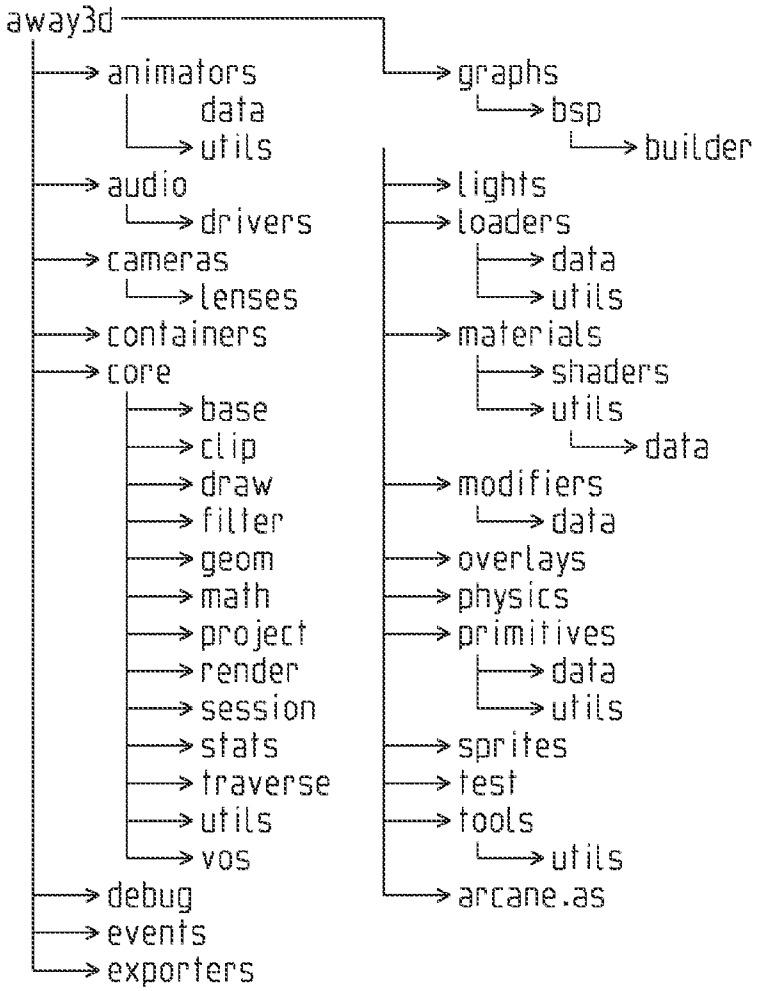

away3d
animators
data
utils
audio
drivers
cameras
lenses
containers
core
base
clip
draw
filter
geom
math
project
render
session
stats
traverse
utils
vos
debug
events
exporters
graphs
bsp
builder
lights
loaders
data
utils
materials
shaders
utils
data
modifiers
data
overlays
physics
primitives
data
utils
sprites
test
tools
utils
arcane.as

Fig. 8

Fig. 9

```
away3d
└──> audio
      ├──> Sound3D.as
      └──> drivers
            ├──> AbstractSound3DDriver.as
            ├──> ISound3DDriver.as
            └──> SimplePanVolumeDriver.as
```

Fig. 10

```
away3d
└──> cameras
      ├──> Camera3D.as
      ├──> HoverCamera3D.as
      ├──> SpringCam.as
      ├──> TargetCamera3D.as
      └──> lenses
            ├──> AbstractLens.as
            ├──> OrthogonalLens.as
            ├──> PerspectiveLens.as
            ├──> SphericalLens.as
            └──> ZoomFocusLens.as
```

Fig. 11

```
away3d
└──> containers
      ├──> Bone.as
      ├──> ILODObject.as
      ├──> IPhysicsScene.as
      ├──> LODObject.as
      ├──> ObjectContainer3D.as
      ├──> Scene3D.as
      └──> View3D.as
```

Fig. 13

```
away3d
└──> debug
      └──> AwayStats.as
```

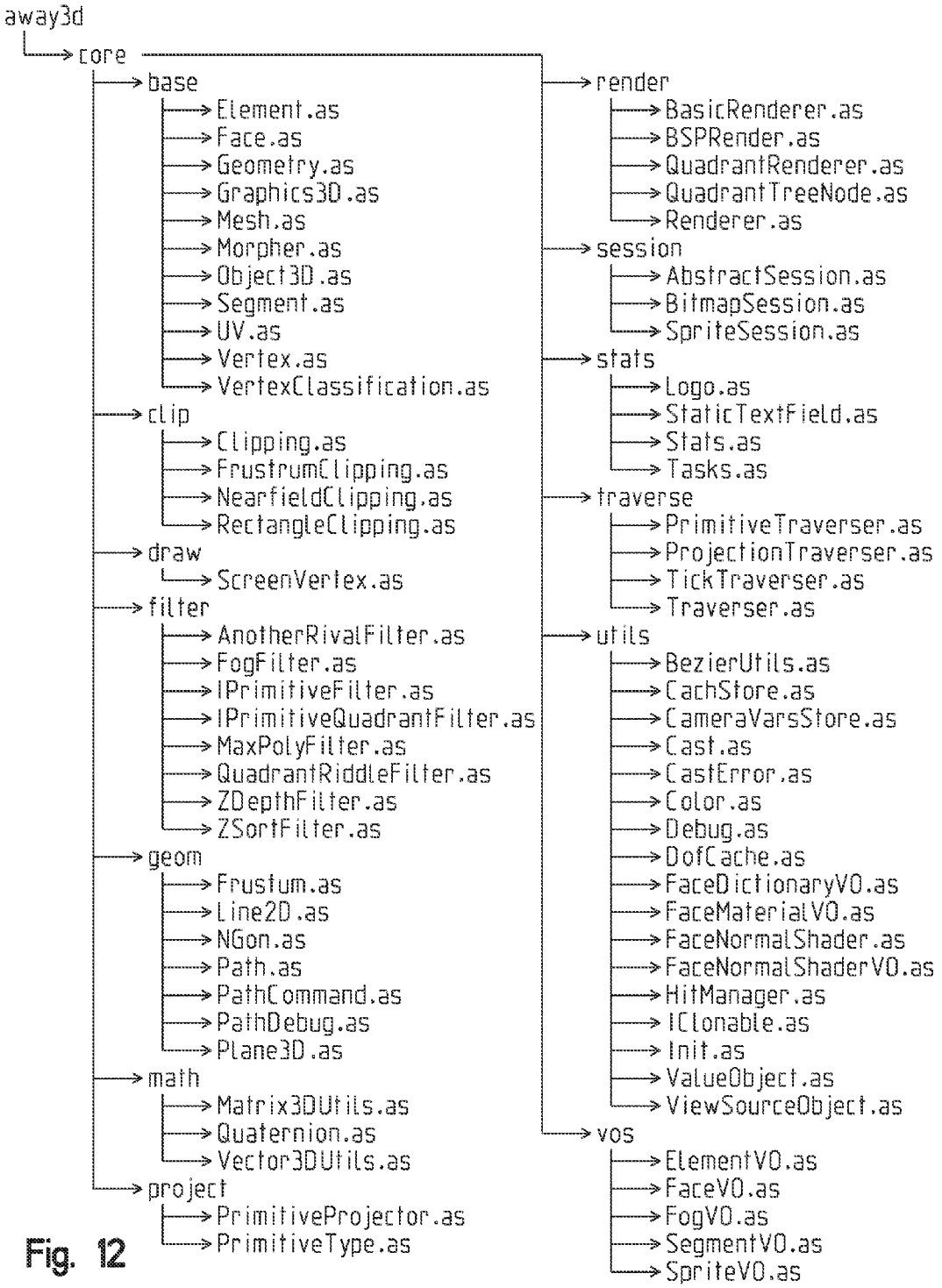

away3d
core
base
Element.as
Face.as
Geometry.as
Graphics3D.as
Mesh.as
Morpher.as
Object3D.as
Segment.as
UV.as
Vertex.as
VertexClassification.as
clip
Clipping.as
FrustrumClipping.as
NearfieldClipping.as
RectangleClipping.as
draw
ScreenVertex.as
filter
AnotherRivalFilter.as
FogFilter.as
IPrimitiveFilter.as
IPrimitiveQuadrantFilter.as
MaxPolyFilter.as
QuadrantRiddleFilter.as
ZDepthFilter.as
ZSortFilter.as
geom
Frustum.as
Line2D.as
NGon.as
Path.as
PathCommand.as
PathDebug.as
Plane3D.as
math
Matrix3DUtils.as
Quaternion.as
Vector3DUtils.as
project
PrimitiveProjector.as
PrimitiveType.as
render
BasicRenderer.as
BSPRender.as
QuadrantRenderer.as
QuadrantTreeNode.as
Renderer.as
session
AbstractSession.as
BitmapSession.as
SpriteSession.as
stats
Logo.as
StaticTextField.as
Stats.as
Tasks.as
traverse
PrimitiveTraverser.as
ProjectionTraverser.as
TickTraverser.as
Traverser.as
utils
BezierUtils.as
CachStore.as
CameraVarsStore.as
Cast.as
CastError.as
Color.as
Debug.as
DofCache.as
FaceDictionaryVO.as
FaceMaterialVO.as
FaceNormalShader.as
FaceNormalShaderVO.as
HitManager.as
IClonable.as
Init.as
ValueObject.as
ViewSourceObject.as
vos
ElementVO.as
FaceVO.as
FogVO.as
SegmentVO.as
SpriteVO.as

Fig. 12

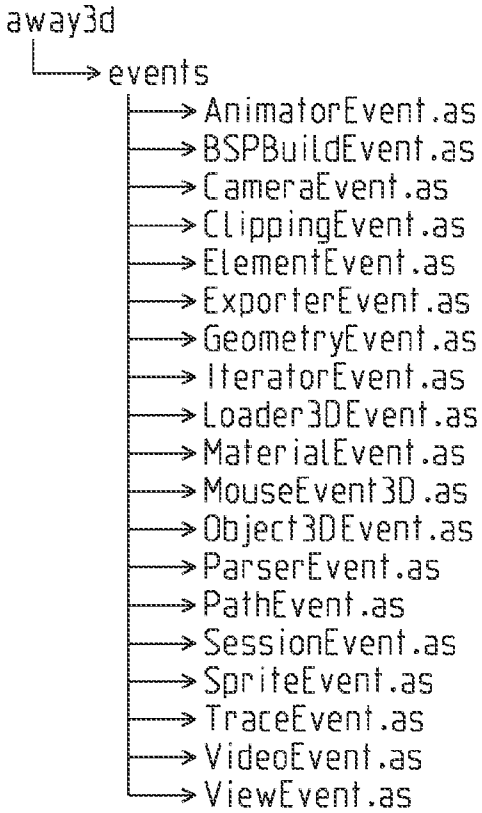

away3d
events
AnimatorEvent.as
BSPBuildEvent.as
CameraEvent.as
ClippingEvent.as
ElementEvent.as
ExporterEvent.as
GeometryEvent.as
IteratorEvent.as
Loader3DEvent.as
MaterialEvent.as
MouseEvent3D.as
Object3DEvent.as
ParserEvent.as
PathEvent.as
SessionEvent.as
SpriteEvent.as
TraceEvent.as
VideoEvent.as
ViewEvent.as

Fig. 14

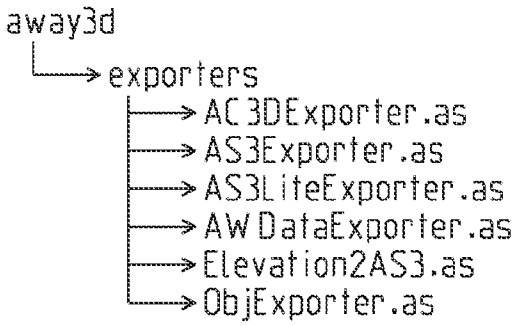

away3d
exporters
AC3DExporter.as
AS3Exporter.as
AS3LiteExporter.as
AWDataExporter.as
Elevation2AS3.as
ObjExporter.as

Fig. 15

```
away3d
└──> extrusions
        ├──> CollisionMap.as
        ├──> Elevation.as
        ├──> ElevationModifier.as
        ├──> ElevationReader.as
        ├──> LatheExtrusion.as
        ├──> LinearExtrusion.as
        ├──> PathDuplicator.as
        ├──> PathExtrusion.as
        ├──> SkinExtrude.as
        └──> TextExtrusion.as
```

Fig. 16

```
away3d
└──> graphs
        ├──> IIterator.as
        ├──> ITreeNode.as
        ├──> TreeIterator.as
        ├──> VectorIterator.as
        └──> bsp
                ├──> BSPHitTest.as
                ├──> BSPMeshManager.as
                ├──> BSPNode.as
                ├──> BSPPortal.as
                ├──> BSPTree.as
                └──> builder
                        ├──> AbstractBuilderDecorator.as
                        ├──> BSPBuilder.as
                        ├──> BSPCollisionPlaneBuilder.as
                        ├──> BSPGeometryBuilder.as
                        ├──> BSPPortalBuilder.as
                        ├──> BSPPVSBuilder.as
                        ├──> BSPTJunctionFixer.as
                        ├──> IBSPBuilder.as
                        ├──> IBSPPlanePicker.as
                        ├──> IBSPPortalProvider.as
                        └──> SimplePlanePicker.as
```

Fig. 17

Fig. 18

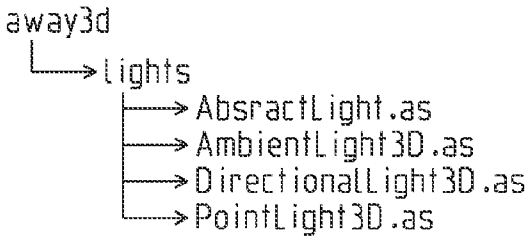
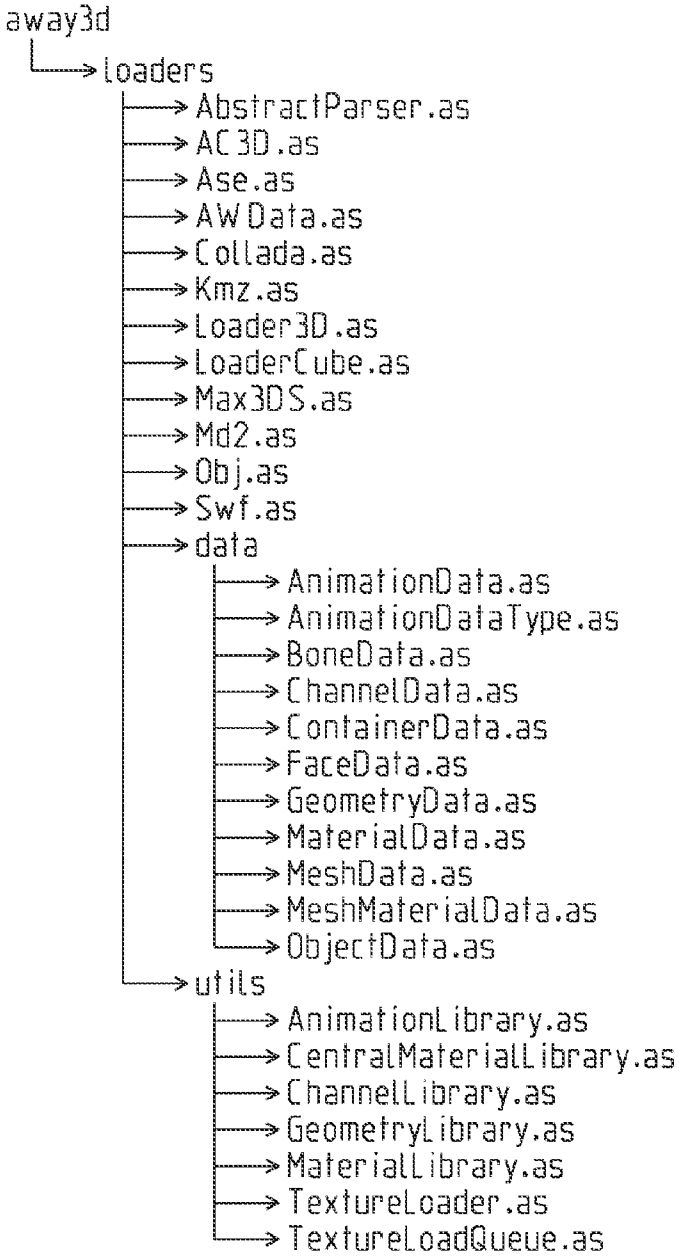

away3d
Lights
AbsractLight.as
AmbientLight3D.as
DirectionalLight3D.as
PointLight3D.as

Fig. 19


away3d
loaders
AbstractParser.as
AC3D.as
Ase.as
AWData.as
Collada.as
Kmz.as
Loader3D.as
LoaderCube.as
Max3DS.as
Md2.as
Obj.as
Swf.as
data
AnimationData.as
AnimationDataType.as
BoneData.as
ChannelData.as
ContainerData.as
FaceData.as
GeometryData.as
MaterialData.as
MeshData.as
MeshMaterialData.as
ObjectData.as
utils
AnimationLibrary.as
CentralMaterialLibrary.as
ChannelLibrary.as
GeometryLibrary.as
MaterialLibrary.as
TextureLoader.as
TextureLoadQueue.as

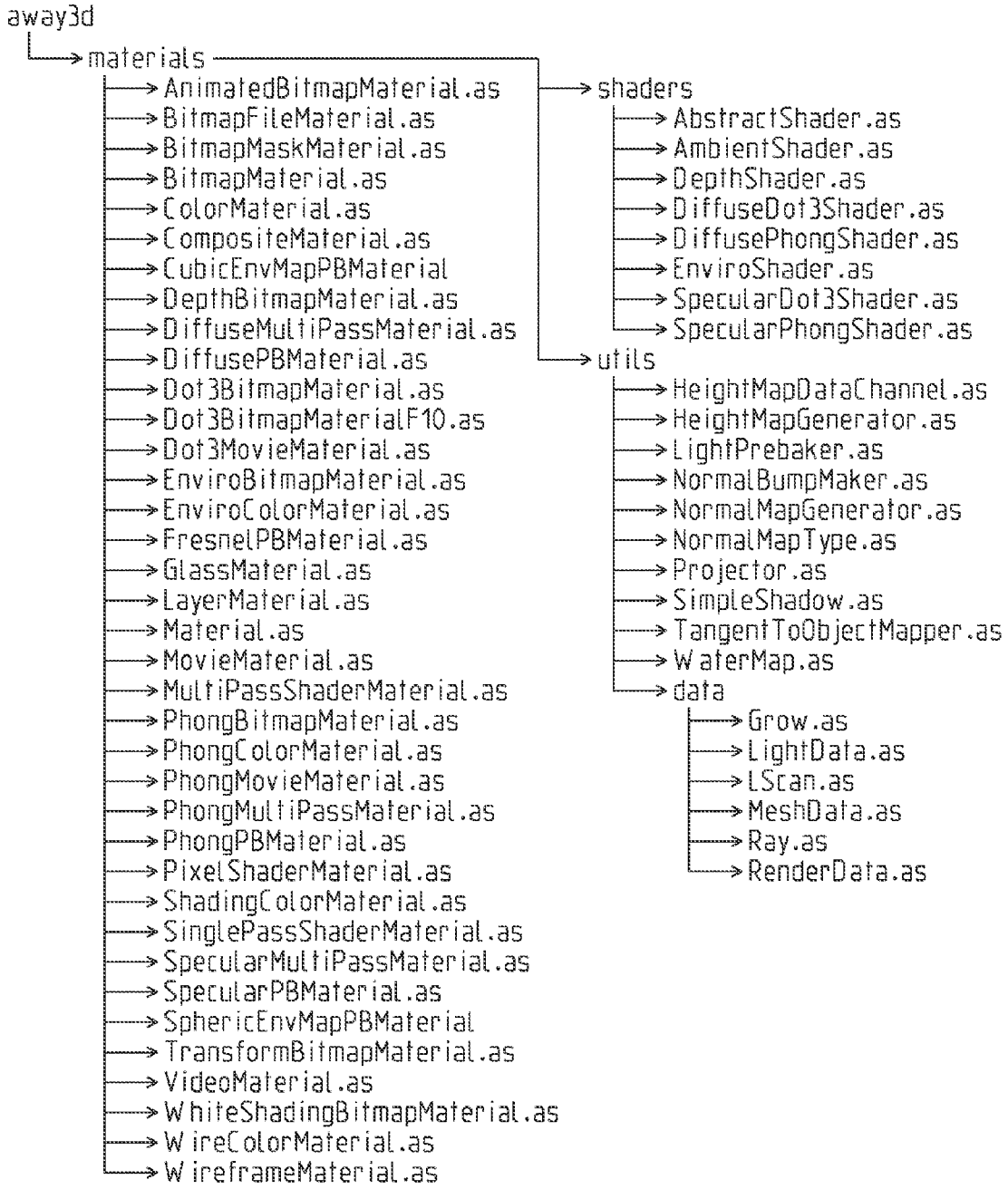

away3d
materials
AnimatedBitmapMaterial.as
BitmapFileMaterial.as
BitmapMaskMaterial.as
BitmapMaterial.as
ColorMaterial.as
CompositeMaterial.as
CubicEnvMapPBMaterial
DepthBitmapMaterial.as
DiffuseMultiPassMaterial.as
DiffusePBMaterial.as
Dot3BitmapMaterial.as
Dot3BitmapMaterialF10.as
Dot3MovieMaterial.as
EnviroBitmapMaterial.as
EnviroColorMaterial.as
FresnelPBMaterial.as
GlassMaterial.as
LayerMaterial.as
Material.as
MovieMaterial.as
MultiPassShaderMaterial.as
PhongBitmapMaterial.as
PhongColorMaterial.as
PhongMovieMaterial.as
PhongMultiPassMaterial.as
PhongPBMaterial.as
PixelShaderMaterial.as
ShadingColorMaterial.as
SinglePassShaderMaterial.as
SpecularMultiPassMaterial.as
SpecularPBMaterial.as
SphericEnvMapPBMaterial
TransformBitmapMaterial.as
VideoMaterial.as
WhiteShadingBitmapMaterial.as
WireColorMaterial.as
WireframeMaterial.as
shaders
AbstractShader.as
AmbientShader.as
DepthShader.as
DiffuseDot3Shader.as
DiffusePhongShader.as
EnviroShader.as
SpecularDot3Shader.as
SpecularPhongShader.as
utils
HeightMapDataChannel.as
HeightMapGenerator.as
LightPrebaker.as
NormalBumpMaker.as
NormalMapGenerator.as
NormalMapType.as
Projector.as
SimpleShadow.as
TangentToObjectMapper.as
WaterMap.as
data
Grow.as
LightData.as
LScan.as
MeshData.as
Ray.as
RenderData.as

Fig. 20

Fig. 21

```
away3d
└──> modifiers
      ├──> HeightMapModifier.as
      ├──> PathAlignModifier.as
      └──> data
            └──> VertexData.as
```

Fig. 22

```
away3d
└──> overlays
      ├──> IOverlay.as
      └──> LensFlare.as
```

Fig. 23

```
away3d
└──> physics
      └──> BSPCollider.as
```

Fig. 25

```
away3d
└──> sprites
      ├──> DepthOfFieldSprite.as
      ├──> DirectionalSprite.as
      ├──> MovieClipSprite.as
      ├──> Sprite3D.as
      └──> SpriteAlign.as
```

Fig. 26

```
away3d
└──> test
      ├──> BaseDemo.as
      ├──> Button.as
      ├──> Panel.as
      ├──> SimpleView.as
      └──> Slide.as
```

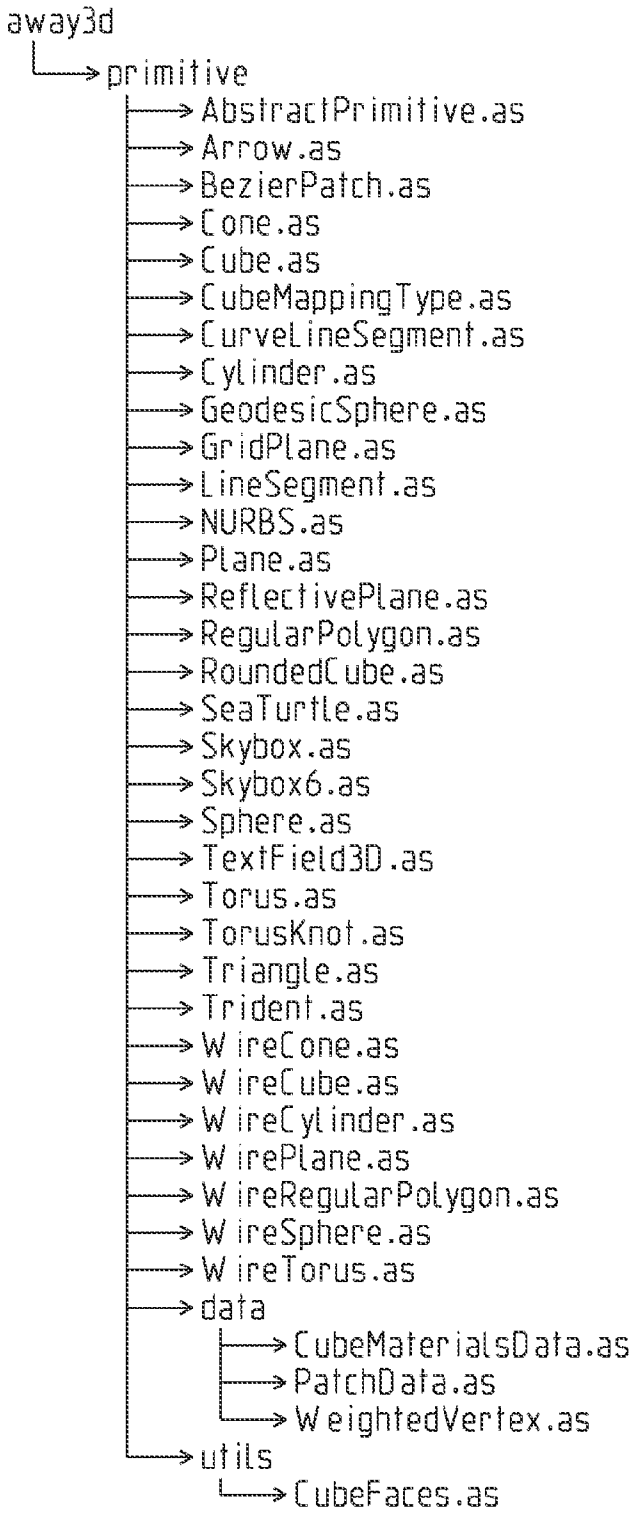

away3d
primitive
AbstractPrimitive.as
Arrow.as
BezierPatch.as
Cone.as
Cube.as
CubeMappingType.as
CurveLineSegment.as
Cylinder.as
GeodesicSphere.as
GridPlane.as
LineSegment.as
NURBS.as
Plane.as
ReflectivePlane.as
RegularPolygon.as
RoundedCube.as
SeaTurtle.as
Skybox.as
Skybox6.as
Sphere.as
TextField3D.as
Torus.as
TorusKnot.as
Triangle.as
Trident.as
WireCone.as
WireCube.as
WireCylinder.as
WirePlane.as
WireRegularPolygon.as
WireSphere.as
WireTorus.as
data
CubeMaterialsData.as
PatchData.as
WeightedVertex.as
utils
CubeFaces.as

Fig. 24

```
away3d
└──> tools
      ├──> Align.as
      ├──> Explode.as
      ├──> Grid.as
      ├──> Merge.as
      ├──> Mirror.as
      ├──> Replicate.as
      ├──> Weld.as
      └──> utils
            ├──> BaryCentricTest.as
            └──> Drag3D.as
```

Fig. 27

```
controller
├──> FLIRServiceStartupCommand.as
└──> NemaParseCommand.as
```

Fig. 28

```
package controller
    import org.robotlegs.mvcs.Command;
    import services.FLIRService;
    public class FLIRServiceStartupCommand extends Command
        [Inject]
        public var flirService:FLIRService;
        override public function execute():void
            flirService.startService();
```

Fig. 29

```
package controller

      import events.AzUpdateEvent;
      import events.CameraUpdateEvent;
      import events.ElUpdateEvent;
      import events.FOVEvent;
      import events.FeedbackMessage;
      import events.NemaUpdateEvent;
      import events.TargetUpdateEvent;
      import model.Interface3Model;
      import org.robotlegs.mvcs.Command;
      public class NemaParseCommand extends Command


            [Inject]
            public var nemaUpdate:NemaUpdateEvent;
            [Inject]
            public var interfaceModel:Interface3Model;

            override public function execute():void

                  var messageString:String = nemaUpdate.nema;
                  //Decipher the type of string and send it to the
                  //correct parser


            private function parseSSS(msg:String):void

                  //Update interfaceModel with the parts of the Camera
                  //Status, aka systemMode, bitStatus, trackerMode &
                  //videoSensor. Then send videoSensor to
                  //CamerUpdateEvent.


            private function parseSPD(msg:String):void

                  //Update interfaceModel with azimuth and elevation.
                  //Then send azimuth to AzUpdateEvent and elevation to
                  //ElUpdateEvent.


            private function parseVSD(msg:String):void

                  //convert the FLIR FOV scale to a 0-1 scale
                  //pass the fov to the model


            private function parseRNG(msg:String):void

                  //pass the range to the model.
                  //send range to TargetUpdateEvent
```

Fig. 30

```
events
  ├──> AzUpdateEvent.as
  ├──> CameraUpdateEvent.as
  ├──> ElUpdateEvent.as
  ├──> FeedbackMessage.as
  ├──> FOVEvent.as
  ├──> NemaUpdateEvent.as
  └──> TargetUpdateEvent.as
```

Fig. 31

```
package events

    import flash.events.Event;
    public class AzUpdateEvent exends Event

        public static const AZIMUTH_UPDATE:STRING="AZIMUTH_UPDATE";
        private var_azimuth:Number;
        public function get azimuth () :Number

            return_azimuth;

        public function AzUpdateEvent(newAz:Number)

            super(AZIMUTH_UPDATE);
            _azimuth = newAz;
```

Fig. 32

```
package events

    import flash.events.Event;
    public class CameraUpdateEvent extends Event

        public static const UPDATE_CAMERA:String="UPDATE_CAMERA";
        private var _cameraType:String;

        public function get cameraType():String

            return_cameraType;

        public function CameraUpdateEvent(newCamType:String):void

            super(UPDATE_CAMERA);
            _cameraType = newCamType;
```

Fig. 33

```
package events

      import flash.events.Event;
      public class ElUpdateEvent extends Event

            public static const ELEVATION_UPDATE:String =
"ELEVATION_UPDATED";

            private var _elevation:Number;
            public function get elevation () :Number
                  return _elevation;
            public function ElUpdateEvent (newEl:Number)
                  super(ELEVATION_UPDATE);
                  _elevation = newEl;
```

Fig. 34

```
package events

      import flash.events.Event;
      public class FeedbackMessage extends Event

            public static const RANGE_ERROR:String = "RANGE_ERROR";
            private var _message:String;
            public function get message () :String
                  return _message;
            public function FeedbackMessage (message:String)
                  super (type) ;
                  _message = message;
```

Fig. 35

```
package events

      import flash.events.Event;
      public class FOVEvent extends Event

            public static const FOV_UPDATE:String = "FOV_UPDATE";
            private var _fieldofview:Number;

            public function get fieldofview():Number
                  return _fieldofview;
            public function FOVEvent (newFOV:Number)
                  super(FOV_UPDATE);
                  _fieldofview = newFOV;
```

Fig. 36

```
package events

    import flash.events.Event;
    public class NemaUpdateEvent extends Event

        public static const NEMA_UPDATE:String = "NEMA_UPDATE";
        private var _nemaStr:String;

        public function get nema ():String
            return_nemaStr;
        public function NemaUpdateEvent (nemaString:String)
            super(NEMA_UPDATE);
            _nemaStr = nemaString;
```

Fig. 37

```
package events

    import flash.events.Event;
    public class TargetUpdateEvent extends Event

        public static const RANGE_UPDATE:String = "RANGE_UPDATE";
        private var _range:Number;

        public function get range():Number
            return _range;
        public function TargetUpdateEvent(newRange:Number)
            super(RANGE_UPDATE);
            _range = newRange;
```

Fig. 38

```
fonts
└──→ MyClearviewHwy.as
```

Fig. 39

```
package fonts

    import flash.text.Font;
    public class MyClearviewHwy extends Font

        public function MyClearviewHwy()
        // defines the font and format of the font
```

Fig. 40

```
interface3
└──→ InterfaceContext.as
```

Fig. 41

```
package interface3

      import controller.FLIRServiceStartupCommand;
      import controller.NemaParseCommand;
      import events.NemaUpdateEvent;
      import events.TargetUpdateEvent;
      import flash.display.DisplayObjectContainer;
      import model.Interface3Model;
      import org.robotlegs.base.ContextEvent;
      import org.robotlegs.mvcs.Context;
      import services.FLIRService;
      import views.FOVBar;
      import views.FOVBarMediator;
      import views.Gimble3D;
      import views.Gimble3DMediator;
      import views.Interface3Mediator;
      import views.SystemStateMediator;
      import views.SystemStateUI;
      import views.TargetDistanceMediator;
      import views.TargetDistanceView;
      import views.TargetUI;
      import views.TargetUIMediator;
      public class InterfaceContext extends Context

            public function
InterfaceContext(contextView:DisplayObjectContainer=null,
autoStartup:Boolean=true)
                  super(contextView, autoStartup);
            override public function startup():void
                  // start by mapping the classes that will be constrained
                  // to Singletons. Map the mediators to the views -
                  // Mediators provide the communication layer between
                  // services/models/commands and the views
                  // in this case, the UI components are being treated as
                  // individual views of the application
```

Fig. 42

```
model
└──→ Interface3Model.as
```

Fig. 43

```
package model

      import org.robotlegs.mvcs.Actor;
      public class Interface3Model extends Actor

////////////////////////////////////////////////////////////////////////
// Complete set of possible data objects that are passed from the FLIR
// az - camera azimuth
// el - camera elevation
// systemMode - the mode the FLIR is currently set to:
// RRATE - Remote Rate Mode
// RPOS - Remote Position Mode
// RHHOLD - Remote Heading Hold
// RAT - Remote AutoTrack Mode
// RSCAN - Remote Scan Mode
// PBIT - Power on BIT Mode
// RATE - Rate Mode (HCU is active)
// POS - Position Mode
// HHOLD - Heading Hold Mode
// AT - Autotrack Mode
// SCAN - Scan Mode
// MNULL - Manual Null Mode
// ANULL - Auto Null Mode
// IBIT - User Initiated Bit Mode
// NUC - NUC Mode
// STBY - Standby Mode
// bitStatus - checks for a bit error
// trackerMode - either TGT (target), or SCN (Scene Mode)
// videoSensor - I (IR), or C (CCD)
// sensorType - I (IR), or C (CCD)
// fov - Field Of View
// currentZoom - reads back on the 1.8x zoom or not (N = normal, or E=
extended)
// range - last range distance obtained from the laser range finder in
meters
////////////////////////////////////////////////////////////////////////
            private var _az:Number;
            private var _el:Number;
            private var _systemMode:String;
            private var _bitStatus:String;
            private var _trackerMode:String;
            private var _videoSensor:String;
            private var _sensorType:String;
            private var _fov:Number;
            private var _currentZoom:Boolean;
            private var _range:Number;

            public function Interface3Model()
                  super();

            ////////////////////////////////////////   GETTERS
            public function get az():Number
                  return _az;
```

```
public function get el():Number
      return _el;
public function get systemMode():String
      return _systemMode;
public function get bitStatus():String
      return _bitStatus;
public function get trackerMode():String
      return _trackerMode;
public function get videoSensor():String
      return _videoSensor;
public function get sensorType():String
      return _sensorType;
public function get fov():Number
      return _fov;
public function currentZoom():Boolean
      return _currentZoom;
public function get range():Number
      return _range;

//////////////////////////////////  SETTERS
public function set az(num:Number):void
      _az = num;
public function set el(num:Number):void
      _el = num;
public function set systemMode(mode:String):void
      _systemMode = mode;
public function set bitStatus(status:String):void
      _bitStatus = status;
public function set trackerMode(mode:String):void
      _trackerMode = mode;
public function set videoSensor(kind:String):void
      _videoSensor = kind;
public function set sensorType(type:String):void
      _sensorType = type;
public function set fov(view:Number):void
      _fov = view;
public function set currentZoom(zoom:Boolean):void

      switch(zoom)
            case "N":
                  _currentZoom = false;
                  break;
            case "E":
                  _currentZoom = true;
                  break;
            default:
                  _currentZoom = false;


public function set range(range:Number):void
      _range = range;
```

Fig. 44

```
services
  ├──→ CameraService.as
  └──→ FLIRService.as
```

Fig. 45

```
package services

      import flash.media.Camera;
      import flash.media.Video;

      import org.robotlegs.mvcs.Actor;
      import flash.display.Sprite;

      public class CameraService extends Sprite

            private var _camera:Camera;
            private var _cameraWidth:Number = 640;
            private var _cameraHeight:Number = 480;
            private var _vid:Video;
            private var _bandwidth:int = 16384; // maximum bandwidth in
bytes per second -0 is variable bandwidth
            private var _quality:int = 50; // 0 to 100 - 0 is a variable
quality amount to maintain framerate
            private var _fps:int = 30;

            public function CameraService()

                  _camera = Camera.getCamera();
                  trace("Camera Name: " + _camera.name);
                  //_camera.setQuality(_bandwidth, _quality);
                  //_camera.setMode(_cameraWidth, _cameraHeight, _fps);

                  _vid = new Video(_cameraWidth, _cameraHeight);
                  _vid.attachCamera(_camera);
                  addChild(_vid);
```

Fig. 46

```
package services

      import events.NemaUpdateEvent;
      import flash.events.Event;
      import flash.events.IOErrorEvent;
      import flash.events.ProgressEvent;
      import flash.events.SecurityErrorEvent;
      import flash.net.Socket;
      import flash.utils.ByteArray;
      import org.robotlegs.mvcs.Actor;
      public class FLIRService extends Actor

            private var _flirSocket:Socket;
            private var _hostName:String = "192.168.42.203";
      private var _portNum:uint = 4002;
            private var _connectionOpen:Boolean = false;

            // create the byte array that will store the socket data
            private var _flirByteAr:ByteArray;

            private function FLIRService()
            private function startService():void
                  // initializes the socket service, create the socket
                  // listeners and connect to the socket
            private function OnSocketConnection(evt:Event):void
                  // sets the system commands
            private function OnSocketResponse(evt:ProgressEvent):void
                  // read and parse the byte array.
                  // once the byte array is handed to be parsed, clear it
                  // to avoid repeating bytes

            private function onSocketClose(evt:Event):void
                  // close the connection, flush the socket, and
                  // clear the byte array

// setFlirESS establishes the frequency of system status updates
// all codes are NEMA strings followed by carriage return '\r' and new
line '\n\'
// must set ESS-Enable System Status with a rate of status updates
// must set ESD-Enable Sensor Data, sends the sensor states
// must set EPD-Enable SGA Position Data, send position status updates
//
////////////////////////////////////////////////////////////////////////
            private function setFlirSystemCommands():void
                        // put the FLIR into remote mode
                        // start with ESS and EPD then ESD

                  private function cancelFlirSystemCommands():void

                  protected function parseSocketData():void
                   // takes the byteArray and checks it for valid nema
                   // strings
```

Fig. 47

```
views
├──> ArrowIndicator.as
├──> Boat.as
├──> Compass.as
├──> FOVBar.as
├──> FOVBarMediator.as
├──> Gimble3D.as
├──> Gimble3DMediator.as
├──> GimbleUI.as
├──> Interface3Mediator.as
├──> Reticle.as
├──> SystemStateMediator.as
├──> SystemStateUI.as
├──> TargetDistanceMediator.as
├──> TargetDistanceView.as
├──> TargetUI.as
└──> TargetUIMediator.as
```

Fig. 48

```
package views

    import flash.display.Sprite;
    public class ArrowIndicator extends Sprite

        private var _arrow:Sprite;
        public function ArrowIndicator()
            // Draws the arrow
```

Fig. 49

```
package views

    import flash.display.Sprite;
    public class Boat extends Sprite

        public function Boat()
            // Draws the boat icon.
```

Fig. 50

```
package views

    import away3d.core.utils.Init;
    import com.greensock.TweenLite;
    import com.greensock.TweenMax;
    import com.greensock.easing.Strong;
    import com.greensock.events.*;
    import flash.display.MovieClip;
    import flash.display.Shape;
    import flash.display.Sprite;
    import flash.text.Font;
    import flash.text.TextField;
    import flash.text.TextFormat;

    public class Compass extends Sprite

        // create vars to hold the heading information
        private var _currentHeading:int = 0;
        private var _newHeading:int;
        private var _headingX:Number = -48;
        private var _headingY:Number = -25;
        private var _headingText:TextField;
        private var _headingInfo:String;

        // create vars for the true north information
        private var _trueNorth:Sprite;
        private var _trueNorthX:int;
        private var _trueNorthY:int;
        private var _trueNorthText:TextField;
        private var _trueNorthArrowLeft:Sprite;
        private var _trueNorthArrowRight:Sprite;

        // the separator bar from the heading to the true north
        private var _bar:Sprite;

        // create a var for the format of all compass text
        private var _compassTextFormat:TextFormat;
        private var _northTextFormat:TextFormat;
        private var _compassFont:Font;

        // add the var for the base Compass
        private var _compassBase:BaseCompass = new BaseCompass();
        private var _compassTween:TweenMax;

        public function Compass()

            addChild(_compassBase);
            createHeading();
            createTrueNorth();
            createBar();


        protected function createHeading():void
            // creates the heading clip and sets the text format for
```

```
        // the heading information
        // instantiate the arrow and position
        // create the text field and set the format and text
        // position the textField

    private function createBar():void;

        _bar = new Sprite();
        _bar.graphics.linestyle(2, 0xFFFFFF, 1);
        _bar.graphics.moveTo(-15,-8);
        _bar.graphics.lineTo(15,-8);
        addChild(_bar);

    protected function createTrueNorth():void
        //creates the True North Indicator
    protected function createCompassTextFormats():void
    public function updateCompass(heading:int):void

        // handling the dispatched event from the model through
        // the mediator
        // add a listener to alter the heading text based on the
        // animation

    private function onTweenUpdate(event:TweenEvent):void
        // listener to alter heading text based on animation.
```

Fig. 51

```
package views

    import flash.display.Sprite;
    public class FOVBar extends Sprite

        private var _fovBarContainer:Sprite;
        private var _fovBarColor:uint = 0xFCFF00;
        private var _fovMask:Sprite;
        public function FOVBar()
            // initializes and draws the fov Bar.
        public function updateFOVMaskScale(newScale:Number):void

            // readjusts the scale of the FOV Mask and updates the
            // FovBarContainer.
```

Fig. 52

```
package views

      import events.FOVEvent;

      import org.robotlegs.mvcs.Mediator;

      public class FOVBarMediator extends Mediator

            [Inject]
            public var fovView:FOVBar;

            override public function onRegister():void

                  eventMap.mapListener(eventDispatcher,
FOVEvent.FOV_UPDATE, onFOVUpdate, FOVEvent);


            private function onFOVUpdate(evt:FOVEvent):void

                  fovView.updateFOVMaskScale(evt.fieldofview);
```

Fig. 53

```
package views

      import events.AzUpdateEvent;
      import events.ElUpdateEvent;
      import org.robotlegs.mvcs.Mediator;
      public class Gimble3DMediator extends Mediator

            [Inject]
            public var gimble3DView:Gimble3D;
            override public function onRegister():void

            // register the mediator to listen for the Azimuth and
            // Elevation data coming from the socket service


            private function onAzUpdate(evt:AzUpdateEvent):void

                  // updated az received, adjusting gimble view

            private function onElUpdate(evt:ElUpdateEvent):void

                  // updated el received, adjusting gimble view
```

Fig. 55

```
package views

    import away3d.containers.*;
    import away3d.lights.DirectionalLight3D;
    import away3d.lights.PointLight3D;
    import away3d.materials.*;
    import away3d.primitives.*;
    import flash.display.Sprite;
    import flash.events.TimerEvent;
    import flash.geom.Vector3D;
    import flash.ui.Keyboard;
    import flash.utils.Timer;
    public class Gimble3D extends Sprite

        protected var _view:View3D;
        protected var _cube1:Cube;
        protected var _cube2:Cube;
        protected var _torus:Torus;
        protected var _fovTorus:Torus;
        protected var _haloTorus:Torus;
        protected var _sphere:Sphere;
        protected var _dirArrow:Cone;
        protected var _maskCone:Cone;
        protected var _objectContainer:ObjectContainer3D;
        protected var _gimbleLightAbove:PointLight3D;
        protected var _gimbleLightBelow:PointLight3D;
        protected var _fovLight:DirectionalLight3D;
        protected var _az:Number = 0;
        protected var _el:Number = 0;
        private var _posX:int;
        private var _posY:int;
        public function Gimble3D(posX:int, posY:int)
            // sets X,Y, the view, the scene and camera.
        protected function _createView():void
            // create the view and add it to the stage
            // relocate the center point of the view to the center of
            // the stage
        protected function _createScene():void
            // setup the lights - above and below
            // then add the light for the fov
            // add the light to the scene
            // torus for the fov in the exact same place and size as
            // the other but with an orange cast
            // create the mask Cone for the FOV - this will be
            // controlled by the Zoom function of the FLIR camera
            // trying a torus to create the halo around the gimble
            // add the indicator arrow
            // assign the scene to the view
        protected function _createCamera():void
        public function updateAz(azimuth:Number):void
        public function updateEl(elevation:Number):void
```

Fig. 54

```
package views

    import flash.events.TimerEvent;
    import flash.globalization.DataTimeFormatter;
    import flash.text.Font;
    import flash.text.TextField;
    import flash.text.TextFormat;
    import flash.utils.Timer;

    public class GimbleUI extends BaseGimble

        private var _craftTime:String;
        private var _craftDateTimeFormat:String;
        private var _craftDate:String;
        private var _craftTimeTimer:Timer;
        private var _craftTimeText:TextField;
        private var _craftDateText:TextField;
        // GimbleUI has 3 sets of text data for the craft
        // information: time/date, lat, and long
        private var _craftInfo:TextField;
        private var _craftInfoTextFormat:TextFormat;
        private var _craftInfoFont:Font;
        private var _craftLat:String;
        private var _craftLong:String;

        public function GimbleUI()

            // Initializes the time format and the timer.

        protected function createCraftInfo():void

            // Builds the text box for the following craft info.

        protected function createCraftInfoFormat():void

            // aligns and sets the font, color and size of the
            // following craft info.

        public function setCraftTime(event:TimerEvent):String


            // sets the date and time based on the selected format
            // for the following craft.

        public function setCraftDateTimeFormat(format:String):String

            // set the format for date and time for the following
            // craft.
```

Fig. 56

```
package views

      import events.TargetUpdateEvent;
      import org.robotlegs.mvcs.Mediator;
      public class Interface3Mediator extends Mediator

            [Inject]
            public var interface3View:Interface3Rebuild;
            public ovverride function onRegister():void

                  eventMap.mapListener(eventDispatcher,
TargetUpdateEvent.RANGE_UPDATE, onRangeUpdate, TargetUpdateEvent);

            private function onRangeUpdate(evt:TargetUpdateEvent):void
                  interface3View.onTargetUpdate();
```

Fig. 57

```
package views

      import flash.display.Sprite;

      public class Reticle extends ReticleBase

            public function Reticle()

                  // draws the reticle
```

Fig. 58

```
package views

      import events.CameraUpdateEvent;
      import org.robotlegs.mvcs.Mediator;

      public class SystemStateMediator extends Mediator

            [Inject]
            public var systemStateView:SystemStateUI;
            override public function onRegister():void

                  eventMap.mapListener(eventDispatcher,
CameraUpdateEvent.UPDATE_CAMERA, onUpdateCamera, CameraUpdateEvent);


            private function onUpdateCamera(evt:CameraUpdateEvent):void
                  systemStateView.updateCameraType(evt.cameraType);
```

Fig. 59

```
package views

    import flash.display.Sprite;
    public class SystemStateUI extends Sprite

        private var _uiSpacer:int = 10;
        private var _contrastAutoIcon:ContrastAutoIcon;
        private var _contrastManIcon:ContrastManIcon;
        private var _contrastHistIcon:ContrastHistIcon;
        private var _brightAutoIcon:BrightnessAutoIcon;
        private var _brightManIcon:BrightnessManualIcon;
        private var _cameraCCDIcon:CameraCCDIcon;
        private var _cameraIRIcon:CameraIRIcon;
        private var _cameraLLTVIcon:CameraLLTVIcon;
        private var _cameraBHIcon:CameraBHIcon;
        private var _cameraWHIcon:CameraWHIcon;
        private var _focusAutoIcon:FocusAutoIcon;
        private var _focusManIcon:FocusManIcon;
        private var _gpsIcon:GPSIcon;
        private var _zoomIcon:Sprite;

        // start and end positions for the UI Icons
        private var _contrastIconStartX:int = -.1;
        private var _contrastIconStartY:int = 1.5;
        private var _contrastIconEndX:int;
        private var _contrastIconEndY:int;
        private var _brightIconStartX:int = 58.95;
        private var _brightIconStartY:int = 3.8;
        private var _brightIconEndX:int;
        private var _brightIconEndY:int;
        private var _cameraIconStartX:int = 96.7;
        private var _cameraIconStartY:int = 32.5;
        private var _cameraIconEndX:int;
        private var _cameraIconEndY:int;
        private var _focusIconStartX:int = 110;
        private var _focusIconStartY:int = 64.8;
        private var _focusIconEndX:int;
        private var _focusIconEndY:int;

        public function SystemStateUI ()

            initDefaultUI ();

        private function initDefaultUI ():void

            // Initializes all of the Icons for the camera status.

        public function updateCameraType(type: String):void

            // updates the Icon for the Camera type.
```

Fig. 60

```
package views

      import events.FeedbackMessage;
      import events.TargetUpdateEvent;
      import org.robotlegs.mvcs.Mediator;
      public class TargetDistanceMediator extends Mediator

            [Inject]
            public var targetDistView:TargetDistanceView;
            override public function onRegister():void

                  eventMap.mapListener(eventDispatcher,
TargetUpdateEvent.RANGE_UPDATE, onTargetUpdate, TargetUpdateEvent);

            private function onTargetUpdate(evt:TargetUpdateEvent):void
                  targetDistView.updateRange(evt.range);
```

Fig. 61

```
package views

      import flash.display.Sprite;
      import flash.text.TextField;
      import flash.text.TextFormat;
      import com.greensock.TweenLite;
      import com.greensock.easing.*;
      public class TargetDistanceView extends Sprite

            private var _targetDistance:Number;
            private var _targetDistLargeFormat:TextFormat;
            private var _targetDistHolder:Sprite;
            private var _targetDistTf:TextField;
            public function TargetDistanceView()

                  createTextFormat();
                  createTargetDistanceHolder();

            private function createTextFormat():void
                  // sets the format of the target text box.
            private function createTargetDistanceHolder():void

                  // draws the target text box and the target range box
                  // next to the reticle.

            public function updateRange(range:Number):void

                  // set the target text on the UI and on the large range
                  // box in the center of the screen
```

Fig. 62

```
package views

      import flash.display.Sprite;
      import flash.events.Event;
      import flash.events.TimerEvent;
      import flash.globalization.DateTimeFormatter;
      import flash.text.TextField;
      import flash.text.TextFormat;
      import flash.utils.Timer;
      public class TargetUI extends BaseTarget

      // set up variables to hold target distance, lat long and bearing
            private var _targetDistance:Number;
            private var _targetLat:String = "38° 52' 8.042\" ";
            private var _targetLong:String = "86° 49' 32.0821\" ";
            private var _targetBearing:String = "246°";
            // create vars for text fields - 1 for target dist, target
info (lat long bearing) and timer
            private var _targetDistTextField:TextField;
            private var _targetInfoTextField:TextField;
            private var _targetTimerTextField:TextField;
            // setup the target timer
            private var _targetTimer:Timer;
            // set up the sprite to hold the target distance near reticle
            private var _targetDistHolder:Sprite;
            private var _targetDistTf:TextField;
            private var _targetDistHolderX:int;
            private var _targetDistHolderY:int;
            // set up var for format of text
            private var _targetDistTextFormat:TextFormat;
            private var _targetInfoTextFormat:TextFormat;
            private var _targetDistLargeFormat:TextFormat;
            private var _targetTimerFormat:TextFormat;
            // set up a var to hold boolean for the large target distance
- initially set to false
            private var _targetDistanceShowing:Boolean = false;
            private var _targetTimeCount:Number = 0;
            private var _pastTime:uint;
            private var _targetTimerFormat:DataTimeFormatter;
            private var _targetStartTime:Date;

            public function TargetUI()

                  createTargetText();
                  createTargetDistanceHolder();
                  setupTimers();

            protected function createTargetText():void
                  // creates the text in the target text box.
            protected function createTargetDistanceHolder():void
                  // create the range to target next to the reticle.
            private function createTextFormats():void
                  // set the format for the text.
            public function setupTimers():void
```

```
        public function resetTimers():void
              // reset target timers
        private function onTargetTimer(evt:TimerEvent):void
              // Calculates the on target time.
        public function updateRange(range:Number):void

              // set the target text on the UI and on the large range
box in the center of the screen
```

Fig. 63

```
package views

      import events.FeedbackMessage;
      import events.TargetUpdateEvent;

      import org.robotlegs.mvcs.Mediator;

      public class TargetUIMediator extends Mediator

            [Inject]
            public var targetUIView:TargetUI;

            override public function onRegister():void

                  eventMap.mapListener(eventDispatcher,
TargetUpdateEvent.RANGE_UPDATE, onTargetUpdate, TargetUpdateEvent);


            private function onTargetUpdate(evt:TargetUpdateEvent):void
                  targetUIView.updateRange(evt.range);
```

Fig. 64

```
package

    import com.greensock.*;
    import com.greensock.easing.*;

    import flash.desktop.NativeApplication;
    import flash.display.MovieClip;
    import flash.display.NativeWindow;
    import flash.display.NativeWindowInitOptions;
    import flash.display.Sprite;
    import flash.display.Stage;
    import flash.display.StageAlign;
    import flash.display.StageDisplayState;
    import flash.display.StageScaleMode;
    import flash.display.KeyboardEvent;
    import flash.media.Camera;
    import flash.media.Video;

    import interface3.InterfaceContext;

    import services.CameraService;

    import views.Compass;
    import views.FOVBar;
    import views.Gimble3D;
    import views.GimbleUI;
    import views.Reticle;
    import views.SystemStateUI;
    import views.TargetDistanceView;
    import views.TargetUI;


    public class Interface3Rebuild extends Sprite

        // establish the RL context
        protected var _interfaceContext:InterfaceContext;

        ////////////////////////////////////////  UI Elements
    // create the different types of view UI components
        // gimbleUI - gimble UI background element
        // gimble - the actual 3D gimble drawn using Away3D
        // compass - the round compass on bottom right
        // reticle - the normal white version of the reticle
        // reticleLock - the red-ringed reticle
        // targetUI - background UI elements for target data

    ////////////////////////////////////////////////////////////

        private var _gimbleUI:GimbleUI;
        private var _gimble:Gimble3D;
        private var _compass:Compass;
        private var _reticle:Reticle;
        private var _reticleLock:ReticleLock;
        private var _targetUI:TargetUI;
```

```
            private var _fovBar:FOVBar;
            private var _targetDistance:TargetDistanceView;

            // variables to place the UI elements to animate in
            private var _compassStartX:Number;
            private var _gimbleBaseStartX:Number;
            private var _compassEndX:Number;
            private var _gimbleBaseEndX:Number;

            private var _video:CameraService;

            private var _camera:Camera;
            private var _cameraWidth:Number;
            private var _cameraHeight:Number;
            private var _vid:Video;
            private var _bandwidth:int = 16384; // maximum bandwidth in
bytes per second - 0 is variable bandwidth
            private var _quality:int = 50; // 0 to 100 - 0 is a variable
quality amount to maintain framerate
            private var _fps:int = 30;

            private var _appScale:int;

            public function Interface3Rebuild ()

                  _interfaceContext = new InterfaceContext (this);
                  init ();
                  initVideo ();
                  buildUI ();


            private function init () :void

            // setup the native application window settings

            // FB is opening 2 windows on startup - so I close the first
default window

            // instantiate the native window and set the size

            // activate the native window and add it to the display list

            // set the stage modes and settings

            // app scale is built in to allow for the UI objects to be
scaled in 4:3 proportion for larger screen sizes


            private function buildUI () :void

            // add the gimble3D elements - takes a width and height as
position elements
            // attempts to position it just right of the side of the left
edge and just above the baseline
```

```
            // gimble UI element - this should be placed to cut off some
of the edges
            // position the gimble base

            // compass - positioned on the bottom right side of the screen

            // target UI - this is initially hidden, since it is triggered
when target lock is acquired

            // reticle - centered on the stage
            // reticle and reticle lock are not actually controlled by
anything
            // these represent the track area and track lock of the FLIR,
but that information is not passed through from the CCFLIR

            // target distance holder - large format range indicator

            // add the UI elements to the display list

            // set the tweens and start them


            private function initVideo () :void
                  // initialize the camera


            public function onTargetUpdate () :void

                  if(!_targetUI.visible)
                        _targetUI.visible = true;
```

Fig. 65

INTUITIVE INTERFACE INITIATIVE

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application claims priority to U.S. Provisional Patent Application Ser. No. 61/637,499, filed Apr. 24, 2012, entitled "INTUITIVE INTERFACE INITIATIVE," the disclosure of which is expressly incorporated by reference herein.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] The invention described herein includes contributions by one or more employees of the Department of the Navy made in performance of official duties and may be manufactured, used, and licensed by or for the United States Government for any governmental purpose without payment of any royalties thereon.

FIELD OF THE INVENTION

[0003] The field of the invention is related to interface systems. In particular, embodiments of the invention are related to standard graphical user interface (GUI) that is compatible with a variety of sensor and information systems such as, for example, Electro Optical and Sensory Equipment designed around the needs of the ends user.

BACKGROUND OF THE INVENTION

[0004] There is a need for a GUI video overlay to provide users the ability to use original equipment manufacturers (OEM) generated GUIs. OEM purchasers utilize a vast and varying inventory of electro-optic (EO) devices for every service and mission. It has proven most cost effective to utilize contractor equipment to fulfill the requirement of the sponsors and operators. Each OEM develops unique methods and implementations of displaying necessary system information to the user. Sponsors, users, or government representatives rarely dictate the design, format or characteristics of required display information. Usually only specific data required by the operator to effectively complete their mission is captured in requirements, but not the placement and characteristics of this information.

[0005] Operational effectiveness is affected in many ways including: too much or too little on-screen information; varying needs of the operators based on roles or missions (ex: gunner versus officer in charge); over-engineered overlays causing option overload; design without operator input; text readability and un-intuitive symbology; and minimal or no capability to configure the display for varying mission requirements.

[0006] Existing EO device video interfaces cause multiple training issues. When new EO systems are introduced, there is a learning curve in order to be proficient in the operation of the system due to the GUI. When operators are required to operate multiple, unique EO devices, each has distinct placement and characteristics that, at times, cause confusion. Training curriculum is required to be developed for each unique system. Changes to the display interface require additional funds in order for the OEM to implement required changes.

[0007] During operation within a system of systems environment, multiple hardware displays can be required to be co-located in order to provide the operator with necessary information to complete their objectives. This increases the cost and decreases the effectiveness of information processing required of the operator.

SUMMARY OF THE INVENTION

[0008] The present invention is a system for standardizing data received for display purposes. The device comprises device drivers that are configured to receive data and video from a plurality of sensory devices that the device drivers are configured for. The data is then used to create standardized GUI overlays. The GUI overlays show the data using both symbology and text. The GUI overlays are then combined with the video and displayed in such a way that the data is shown in the same way, independent of the source of the sensory.

[0009] Embodiments of the invention were developed to provide features such as a standardized GUI that is compatible with existing EO and sensory equipment designed around the needs of the end user.

[0010] In certain aspects, the present invention provides a novel device that can effectively display data from a variety of devices in a manner that is standardized and intuitive to use. Accordingly, in one embodiment, the present invention provides a device that includes one or more electro-optical devices, a processor, a plurality of storage media, a display, a software program, a library of device drivers and a video data grabber. The software program is executed by the processor and stored on the storage media. The software program is configured to detect each electro-optical device connected, select a device driver from the library of device drivers which are stored on the storage media, and use that device driver to translate data from the attached electro-optical device. The software program then uses the data that it translated to create a standardized overlay. The overlay is designed to communicate critical information quickly and be intuitive to use. The video data grabber receives video from each electro-optical device and then the software combines the video with the overlay to be shown on the display.

[0011] Additional features and advantages of the present invention will become apparent to those skilled in the art upon consideration of the following detailed description of the illustrative embodiment exemplifying the best mode of carrying out the invention as presently perceived.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] The detailed description of the drawings particularly refers to the accompanying figures in which:

[0013] FIG. **1** shows an overview of hardware and software architecture in accordance with of one embodiment of the invention that allows that data and video from a variety of devices to be collated and sent display units;

[0014] FIG. **2** shows another embodiment of the system including an operator system, a client system, a sensor system, and a video encoder system;

[0015] FIG. **3** shows a GUI display in accordance with another embodiment of the invention;

[0016] FIG. **4** shows a GUI display in accordance with another embodiment of the invention where the camera view shows the camera in an Infrared (IR) Mode;

[0017] FIG. **5** shows a model of a software architecture framework that translates data into a common and configurable feed in accordance with one embodiment of the invention;

[0018] FIG. **6** shows a high-level overview of the software architecture in accordance with one embodiment of the invention;
[0019] FIG. **7** shows a high-level view of the directories of the code of one embodiment of the invention;
[0020] FIG. **8** shows the directory tree of the contents of the away3d directory;
[0021] FIG. **9** shows the contents of the audio directory in the away3d directory;
[0022] FIG. **10** shows the contents of the cameras directory in the away3d directory;
[0023] FIG. **11** shows the contents of the containers directory in the away3d directory;
[0024] FIG. **12** shows the contents of the core directory in the away3d directory;
[0025] FIG. **13** shows the contents of the debug directory in the away3d directory;
[0026] FIG. **14** shows the contents of the events directory in the away3d directory;
[0027] FIG. **15** shows the contents of the exporters directory in the away3d directory;
[0028] FIG. **16** shows the contents of the extrusions directory in the away3d directory;
[0029] FIG. **17** shows the contents of the graphs directory in the away3d directory;
[0030] FIG. **18** shows the contents of the lights directory in the away3d directory;
[0031] FIG. **19** shows the contents of the loaders directory in the away3d directory;
[0032] FIG. **20** shows the contents of the materials directory in the away3d directory;
[0033] FIG. **21** shows the contents of the modifiers directory in the away3d directory;
[0034] FIG. **22** shows the contents of the overlays directory in the away3d directory;
[0035] FIG. **23** shows the contents of the physics directory in the away3d directory;
[0036] FIG. **24** shows the contents of the primitives directory in the away3d directory;
[0037] FIG. **25** shows the contents of the sprites directory in the away3d directory;
[0038] FIG. **26** shows the contents of the test directory in the away3d directory;
[0039] FIG. **27** shows the contents of the tools directory in the away3d directory;
[0040] FIG. **28** shows the contents of the controller directory;
[0041] FIG. **29** shows a representation of the contents of FLIRServiceStartupCommand;
[0042] FIG. **30** shows a representation of the contents of NemaParseCommand;
[0043] FIG. **31** shows the contents of the events directory;
[0044] FIG. **32** shows a representation of the contents of AzUpdateEvent;
[0045] FIG. **33** shows a representation of the contents of CameraUpdateEvent;
[0046] FIG. **34** shows a representation of the contents of ElUpdateEvent;
[0047] FIG. **35** shows a representation of the contents of FeedbackMessage;
[0048] FIG. **36** shows a representation of the contents of FOVEvent;
[0049] FIG. **37** shows a representation of the contents of NemaUpdateEvent;
[0050] FIG. **38** shows a representation of the contents of TargetUpdateEvent;
[0051] FIG. **39** shows the contents of the fonts directory;
[0052] FIG. **40** shows a representation of the contents of MyClearviewHwy;
[0053] FIG. **41** shows the contents of the Interface3 directory;
[0054] FIG. **42** shows a representation of the contents of InterfaceContext;
[0055] FIG. **43** shows the contents of the model directory;
[0056] FIG. **44** shows a representation of the contents of Interface3Model;
[0057] FIG. **45** shows the contents of the services directory;
[0058] FIG. **46** shows a representation of the contents of CameraService;
[0059] FIG. **47** shows a representation of the contents of FLIRService;
[0060] FIG. **48** shows the contents of the views directory;
[0061] FIG. **49** shows a representation of the contents of ArrowIndicator;
[0062] FIG. **50** shows a representation of the contents of Boat;
[0063] FIG. **51** shows a representation of the contents of Compass;
[0064] FIG. **52** shows a representation of the contents of FOVBar;
[0065] FIG. **53** shows a representation of the contents of FOVBarMediator;
[0066] FIG. **54** shows a representation of the contents of Gimble3D;
[0067] FIG. **55** shows a representation of the contents of Gimble3DMediator;
[0068] FIG. **56** shows a representation of the contents of GimbleUI;
[0069] FIG. **57** shows a representation of the contents of Interface3Mediator;
[0070] FIG. **58** shows a representation of the contents of Reticle;
[0071] FIG. **59** shows a representation of the contents of SystemStateMediator;
[0072] FIG. **60** shows a representation of the contents of SystemStateUI;
[0073] FIG. **61** shows a representation of the contents of TargetDistanceMediator;
[0074] FIG. **62** shows a representation of the contents of TargetDistanceView;
[0075] FIG. **63** shows a representation of the contents of TargetUI;
[0076] FIG. **64** shows a representation of the contents of TargetUIMediator; and
[0077] FIG. **65** shows a representation of the contents of Interface3Rebuild.

DETAILED DESCRIPTION OF THE INVENTION

[0078] The embodiments of the invention described herein are not intended to be exhaustive or to limit the invention to precise forms disclosed. Rather, the embodiments selected for description have been chosen to enable one skilled in the art to practice the invention.
[0079] One exemplary system accepts video and data from a wide array of systems utilizing multiple formats. This video and data is then displayed via the GUI. Data transmission formats and characteristics can vary among multiple OEM vendors. The invention may operate with commercially stan-

dard formats that can comprise, for example, 90% of the EO devices. Embodiments of the invention can include a system where power required to operate an exemplary Intuitive Interface Initiative (3I) system would need to be developed around military standards and also would need to operate independently of craft/vehicle power sources. Exemplary GUI embodiments in accordance with an embodiment of the invention are intuitive and easy to interpret. Exemplary embodiments display information unobtrusively to allow effective operation of the EO sensor.

[0080] Hardware architecture required to accomplish 3I objectives is flexible and adaptive to cover the broad spectrum of existing systems. Embodiments of the 3I system also may be upgraded to evolve as technology continues to increase performance of the system.

[0081] One embodiment of the invention uses an object oriented architecture written in Actionscript 3.0 language (AS3). The exemplary framework was written to be modular which allows the code to be updated to add additional devices by writing a "translator" class. This class converts the device's raw data into a format that the software's "parser" class could use and distribute to the device specific objects displayed in the user's GUI. The device would then be added to the framework's library of compatible devices that the software recognizes along with any device specific GUI assets (target reticles, icons, etc.).

[0082] Many advantages are realized from implementation of an example of a 3I system including an increase in effectiveness. Data displays can be designed and tailored to meet different user's needs. Consistency of overlays can be realized for a wide variety of systems and sensors. Response times to interpret and act on data can be reduced. Training costs and curriculum can be standardized and on the job training reduced resulting in minimal cost for changes to an exemplary overlay as opposed to utilizing OEM contractors to implement. As new systems are fielded to replace obsolete or ineffective systems, minimal impacts will be realized by the operators related to an exemplary video overlay. As video is ported off the EO system to be displayed by other users, multiple devices can be used. Screen over clutter can be tailored and reduced to meet the needs of the operator. Multiple sensors can feed data to a common display reducing operator workload and duplicative display costs. Customization can be realized by operators to efficiently get the information needed to complete objectives and thus reducing errors or misinterpretation of the data.

[0083] An embodiment of a **31** system provides capabilities including an ability to make both the video and serial data from a common off-the-shelf (COTS) camera available to a single application. In this example, combinations of hardware for conversion and type of code chosen (e.g., AS3) allows for the capabilities for 3I's utilization in many types of operating systems and devices.

[0084] An embodiment of 3I can be modified and used for any system that receives data and displays to an operator. The system could adapt to provide individual overlays for sensors on radars, lethal effectors, less than lethal effectors, global positioning systems, other visual augmentation systems, and any other military hardware that provides a video source or interface with information displayed on the screen.

[0085] Referring to FIG. **1**, an exemplary overview of the hardware and some aspects of software architecture of the Translator Device **10** is shown. An assembly **31** is shown with a variety of inputs, outputs, and internal components. Translator Device **10** has Data Ports **1**, which allow Translator Device **10** to receive data feeds from Sensor Devices **5**. Data received through the Data Ports **1** is then sent to the Software **13**. Within the Software, interface software, such as for example, Modular Mission Payload Architecture (MMPA) Software **11**, provides an interface between a sensor, e.g., Sensor Device **5**, and other software systems such as Operator Monitor **26** or Client Systems **28**. In this example, MMPA Software **11** detects which Sensor Device **5** attached to Data Port **1**. MMPA Software **11** then selects the appropriate Device Driver **14***a*-**14***c* to use with the Sensor Device **5**. Then the selected Device Driver **14***a*-**14***c* is used to convert the data from the Sensor Device **5** into a standardized format. The data is then sent to Flash Software **12**. Flash Software **12** uses that data to create text and symbology overlays.

[0086] The Translator Device **10** also has Video-in Ports **3** which allows the Translator Device **10** to read video data from Sensor Devices **5**. This video information is then sent to a Software Development Kit (SDK) Frame Grabber **9** and a Hardware Encoder **7**. SDK Frame Grabber **9** combines the video with the text and symbology overlay from the Flash Software **12** and sends the combined data and video to Video Out Port **24**. Video Out Port **24** is connected to one or more Operator Monitor **26** to display the combined information.

[0087] In this example, Hardware Encoder **7** converts the red, green, blue (RGB) video to H.264. The H.264 data is then combined with the text and symbology overlay from the flash software **12** and sent to Ethernet Switch **16**. This allows the information to be sent out through a RJ45 Port **20** to Client Systems **28**. Alternatively, information may be sent out via a Wireless Transmitter **18** which is connected to an Antenna Jack **22** to transmit the information to be picked up by Mobile Client Systems **30**.

[0088] FIG. **2** details an exemplary **31** architecture with one sensor type, e.g., forward looking infrared (FLIR) laser camera software classes. Referring to FIG. **2**, an exemplary EO Device **40** is connected to an Operator/Gunner Configuration system **44**. EO Device **40** sends data and video to a Ruggedized Mobile Computing Device, e.g., a Personal Computer (PC) **52**, which transmits data to Monitor **54** for display. Rugged PC **52** also receives information from a Controller **56** which allows a user to control where EO Device **40** is aimed. Video from the EO device **40** is also sent to a Video Controller **42** which sends video data to a Client Configuration System **50**. This exemplary Client Configuration System **50** allows data to be sent through Wireless Network **58** to be displayed on Mobile Devices **60**. Client Configuration System **50** also allows data to be sent via Wired Network **62** for display on Computers **64**.

[0089] FIGS. **3** and **4** show different potential embodiments of a standardized GUI. In these example embodiments, the combined video information and symbology overlays are displayed in the User View **200**. In User View **200**, a Targeting Reticule **210** is provided which is located over a video image of a Target **205**. Targeting Reticle **210** also displays the range to Target **205**. An exemplary GUI also contains a Target Text Box **220** which displays Target's current position in latitude and longitude, and bearing and distance from the Sensory Device **5**.

[0090] Following Craft Text Box **240** displays information about a device associated with the GUI, e.g., EO Device **40**, including location in latitude and longitude and date and time the video is captured. An Orientation Element **260** displays a graphical representation of a gyroscopic orientation of a

device associated with an exemplary GUI, e.g., direction of a field of view of a sensor associated with an EO Device **40**. Compass **230** further displays the direction of orientation of the EO Device **40**. Camera Status Icons **250** allows a user to quickly view the state of the EO Device **40**. Camera Status Icons **250** update with regards to the EO Device **40** attached, only displaying those Icons that are available for use with an attached EO Device **40**. The Camera Status Icons **250** can include Contrast **251**, Brightness **252**, Camera Mode **253**, and Focus **254**.

[0091] FIG. **4** shows User View **201** for when a FLIR camera attached and is engaged in Infrared Mode. In this mode, the video feed from the camera is changed. This change in mode is shown by Camera Mode Icon **253** which in FIG. **4** is shown as IR. Also a Magnification Icon **255** may also be used. Unlike the other Camera Status Icons **251-254**, some Icons like Magnification **255** and GPS (not shown) are only visible when those modes are activated or the devices attached support them.

[0092] FIG. **5** shows an exemplary programming interface in accordance with one embodiment of the invention. In particular, FIG. **5** shows exemplary software code organized by classes along with high level communication flow for segments of software code within the Context **53**. In this embodiment the application utilizes the Robotlegs coding architecture and 3rd party plugins/engines. Robotlegs is an Action Script 3 architecture/framework that utilizes automated metadata based dependency injection. The event driven Robotlegs framework made the 3I code modular enough to enable the addition of future functionality with new sensors.

[0093] Context **53** initializes the dependency injection between Mediators (e.g. **83-97**) and Views (e.g. **101-119**). StartupCommand **51** activates FLIRService **55** and NemaParseCommand **63**. FLIRService **55** defines the data socket settings including an internet protocol (IP) address and an assigned computer interface port address, connects to a socket service provided by an operating system on a computer e.g., the PC **52** or client system **28**, and configures the FLIR communication to be transmitted. When a message comes in from FLIR **57**, FLIRService **55** stores the data from a socket in buffer, and parses the socket data. Then NemaUpdateEvent **59** gets socket data as a NMEA string which is based on the NMEA 0183 standard. NemaUpdateEvent **59** sends that string to NemaParseCommand **63** which parses the NMEA string and dispatches events with the parsed data to update the values associated with the parsed values. NemaParseCommand **63** also activates Interface3Model **85**, which gets and sets azimuth, elevation, system mode, bit status, tracker mode, video sensor, sensor type, field of view, current zoom, and range.

[0094] Events associated with NemaParseCommand **63** include AzUpdateEvent **69** (updates azimuth value), ElUpdateEvent **71** (updates elevation value), FOVEvent **73** (updates field of view value), CameraUpdateEvent **75** (updates camera type), TargetUpdateEvent (updates range value) **77**, and FeedbackMessage (returns a message string) **81**.

[0095] FOVBarMediator **87** waits for an update from FOVEvent **73** to update the scale of field-of-view mask in FOVBar **107**. FOVBar **107** then defines, generates and animates the field-of-view components to be displayed.

[0096] Gimble3DMediator **85** waits for an update from AzUpdateEvent **69** to update the azimuth in Gimble3D **105**. Gimble3DMediator **85** also waits for an update from ElUpdateEvent **71** to update the elevation in Gimble3D **105**. Gimble3D **105** defines, generates, and animates the camera graphics to be displayed.

[0097] Interface3Mediator **83** waits for an update from TargetUpdateEvent **77** to update the target range objects. Interface3Mediator **83** also interacts with Interface3 **101** to create GUI objects shown in FIGS. **3** & **4**.

[0098] SystemStateMediator **93** waits for an update from CameraUpdateEvent **75** to update a camera type in SystemStateUI **113**. SystemStateUI **113** defines and generates Camera Status Icons **250**.

[0099] TargetDistanceMediator **97** waits for an update from TargetUpdateEvent **77** to update target reticle range on screen values in TargetDistanceView **117**. TargetDistanceView **117** then defines and generates target reticle range data as seen in close proximity to Targeting Reticle **210**.

[0100] TargetUIMediator **95** waits for an update from TargetUpdateEvent **77** to update the range to the target history data in TargetUI **115**. TargetUI **115** then defines and generates target history data on the lower right side of the screen in Target Text Box **220**.

[0101] GimbleUIMediator **89** interacts with the GimbleUI **109** to define and generate the following craft UI of time, latitude, and longitude as seen in the Following Craft Text Box **240**.

[0102] ReticleMediator **91** interacts with Reticle **111** to display Targeting Reticle **210** on the screen.

[0103] Compass **119** defines, generates, and animates Compass **230**.

[0104] FIG. **6** shows a functional diagram of computer software modules for a system in accordance with one embodiment of the invention. Context **53** initializes dependency injection and other utilities. It is the beginning of the application. Context **53** initializes the Commands **152** and matches Mediators **157** with Views **158**. Commands **152** represent individual actions that the application can perform. Commands **152** initialize Events **154** and Model **157**. In this embodiment, Commands **152** consists of StartupCommand **51** and NemaParseCommand **63**.

[0105] Events **154** are used to pass parameters to event listeners when an event occurs. Events **154** in this embodiment include AzUpdateEvent **69**, ElUpdate Event **71**, FOVEvent **73**, CameraUpdateEvent **75**, TargetUpdateEvent **77**, NemaUpdateEvent **59**, and FeedbackMessage **81**.

[0106] Event, e.g., **154**, are passing parameters, e.g., **81**, to Mediators **157** which in turn are stored in Model **164** and passed to Views **158** by Mediators **157**. Model **164** stores data and represents the current state of the application. Interface3Model **85** stores parameters including azimuth, elevation, system mode, bit status, tracker mode, video sensor, sensor type, field of view, current zoom, and range.

[0107] Mediators **157** manage data between the application view components, Views **158**, and other objects within the application. As shown in FIG. **5**, there are several Mediators which interact with the Views, each keeping track of a different Event **154** and parameter, which is then used to update the appropriate View **158**. Mediators **157** include Interface3Mediator **83**, Gimble3DMediator **85**, FOVBarMediator **87**, GimbleUIMediator **89**, ReticleMediator **91**, SystemStateMediator **93**, TargetUIMediator **95** and TargetDistanceMediator **97**.

[0108] Views **158** are viewable components that contain methods that can be called by the Mediators **157**. Views **158** interact with Images **160** and Away3d **162** to draw the images

and text seen on the screen. Views **158** also retrieve the data from the Model **164** to display on the screen. In this embodiment, Views **158** include ArrowIndicator (defines characteristics of compass arrow generation), Boat (displays boat graphic), Compass **119**, FOVBar **107**, Gimble3D **105**, GimbleUI **109**, Interface3 **101**, Reticle **111**, SystemStateUI **113**, TargetDistanceView **117**, and TargetUI **115**.

[0109] Services **156** are used to communicate with programs and/or devices outside the application to get data for the models. In this embodiment CameraService defines the video stream settings and FLIRService **55** sets up and communicates with the FLIR Camera **57**.

[0110] Images **160** contain basic images such as buttons. Away3d **162** contains the classes that draw on the display and show the video.

[0111] The Robotlegs architecture of this embodiment also includes several plug-ins, including Away3D, TweenLite, and TweenMax. Away3D is an open source 3D engine for Adobe Flash used to render 3D models and perform additional 3D related tasking. Away3D was used in the **31** software to implement 3D models such as the camera and Taurus graphic. TweenLite is a fast, lightweight, tweening engine that handles the movement of graphics and animations. TweenMax is a fast, lightweight, tweening engine with additional capabilities and handles the movement of graphics and animations.

[0112] FIGS. **7-28**, **31**, **39**, **41**, **43**, **45**, and **48** show various views of an exemplary file structure of example code of one embodiment of the invention.

[0113] FIGS. **29** and **30** show representations of the code in the controller directory.

[0114] FIGS. **32-38** show representations of the code in the events directory.

[0115] FIGS. **40**, **42** and **44** show representations of the code in the fonts, interface3 and model directories.

[0116] FIGS. **46-47** show representations of the code in the services directory.

[0117] FIGS. **49-64** show representations of the code in the views directory.

[0118] FIG. **65** shows a representation of the Interface3Rebuild.

[0119] Although the invention has been described in detail with reference to certain preferred embodiments, variations and modifications exist within the spirit and scope of the invention as described and defined in the following claims.

1. A device comprising:

one or more electro-optical devices;

a processor;

a plurality of storage media;

a display showing a graphical user interface;

a software program adapted to produce said graphical user interface;

a library of device drivers;

a video data grabber;

wherein the software program is executed by the processor and stored on the storage media;

wherein said software program is configured to detect each electro-optical device connected, select a device driver from the library of device drivers which are stored on the storage media, and use said device driver to translate data from each electro-optical device;

wherein the software program uses translated data to create a standardized overlay which communicates alpha-numeric and graphical interface information using similar or standardized symbology in said graphical user interface, wherein said symbology is positioned to minimize obstruction of a field of view of a target, said symbology is grouped associated with similar classes of data, said similar classes of data comprise user information, device information, and target information, said minimization of obstruction of said field of view comprising placing at least a portion of similar classes of data on a periphery of said graphical user interface and grouping said similar classes of data together into portions of said periphery comprising a lower left and lower right hand section of said graphical user interface;

wherein said video data grabber receives video from each electro-optical device;

wherein said video is combined with the similar or standardized graphical user interface symbology overlay and shown on the display.

2. A computer program comprising:

a plurality of processing sequences stored on a non-transitory computer readable medium comprising;

wherein a 1st processing sequence is adapted to detect a plurality of device types;

wherein a 2nd processing sequence is adapted to select a device driver from a library of device drivers that is configured for a detected device;

wherein a 3rd processing sequence is adapted to receive data from the detected device;

wherein a 4th processing sequence is adapted to translate data from the detected device;

wherein a 5th processing sequence is adapted to use translated data to create a standardized Graphical User Interface overlay;

wherein a 6th processing sequence is adapted to receive video from the detected device; and

wherein a 7th processing sequence is adapted to combine the video with the standardized Graphical User Interface overlay and display on a monitor;

wherein said 5th processing sequence uses said translated data to create a standardized overlay which communicates alpha-numeric and graphical interface information using similar or standardized symbology in said graphical user interface, wherein said symbology is positioned to minimize obstruction of a field of view of a target, said symbology is grouped associated with similar classes of data, said similar classes of data comprise user information, device information, and target information, said minimization of obstruction of said field of view comprising placing at least a portion of similar classes of data on a periphery of said graphical user interface and grouping said similar classes of data together into portions of said periphery comprising a lower left and lower right hand section of said graphical user interface.

3. A system for interfacing multiple systems in a common user interface, comprising:

a sensor interface processing sequence comprising detecting a sensor connected and selecting a device driver from a plurality of device drivers to interface with the sensor;

a plurality of event update processing sequences comprising a first event processing sequence, second, etc;

wherein said first event processing sequence is adapted to receive and update azimuth information provided by a sensor;

wherein said second event processing sequence is adapted to receive and update elevation information provided by a sensor;

wherein said third event processing sequence is adapted to receive and update field-of-view information provided by a sensor;

wherein said fourth event processing sequence is adapted to receive and update camera type information provided by a sensor;

wherein said fifth event processing sequence is adapted to receive and update range information provided by a sensor;

wherein said sixth event processing sequence is adapted to receive and update error messages provided by a sensor;

a plurality of mediator modules comprising Interface3Mediator, Gimble3DMediator, FOVBarMediator, GimbleUIMediator, ReticleMediator, SystemStateUIMediator, TargetUIMediator and TargetDistanceMediator;

a plurality of graphical user interface processing sequences comprising ArrowIndicator, Compass, FOVBar, Gimble3D, GimbleUI, Interface3, Reticle, SystemStateUI, TargetDistanceView, and TargetUI;

wherein said ArrowIndicator is adapted to define, generate, and animate an orientation element;

wherein said Compass is adapted to define, generate and animate the compass components;

wherein said FOVBarMediator is adapted to wait for an update to field-of-view information and then update the scale of a field-of-view mask in FOVBar which is adapted to define, generate and animate field-of-view components to be displayed;

wherein said Gimble3DMediator is adapted to wait for updates to azimuth and elevation information and then update the azimuth and elevation in Gimble3D which is adapted to define, generate, and animate camera graphics to be displayed;

wherein said GimbleUIMediator is adapted to interact with the GimbleUI which is adapted to define and generate following craft data;

wherein said Interface3Mediator is adapted to wait for an update range information and interacts with Interface3 which is adapted to create GUI objects displayed;

wherein said ReticleMediator is adapted to interact with Reticle to display a targeting reticle;

wherein said SystemStateMediator is adapted to wait for an update to camera type information and then to update camera type in SystemStateUI which is adapted to define and generate Camera Status Icons;

wherein said TargetDistanceMediator is adapted to wait for an update to range information and then to update range values in TargetDistanceView which is adapted to define and generate target reticle range data ;

wherein said TargetUIMediator is adapted to wait for an update to range information and then to update the range information in TargetUI which is adapted to define and generate target history data .

* * * * *