

# **Building Security Simulator**

## **Architecture Design Document**

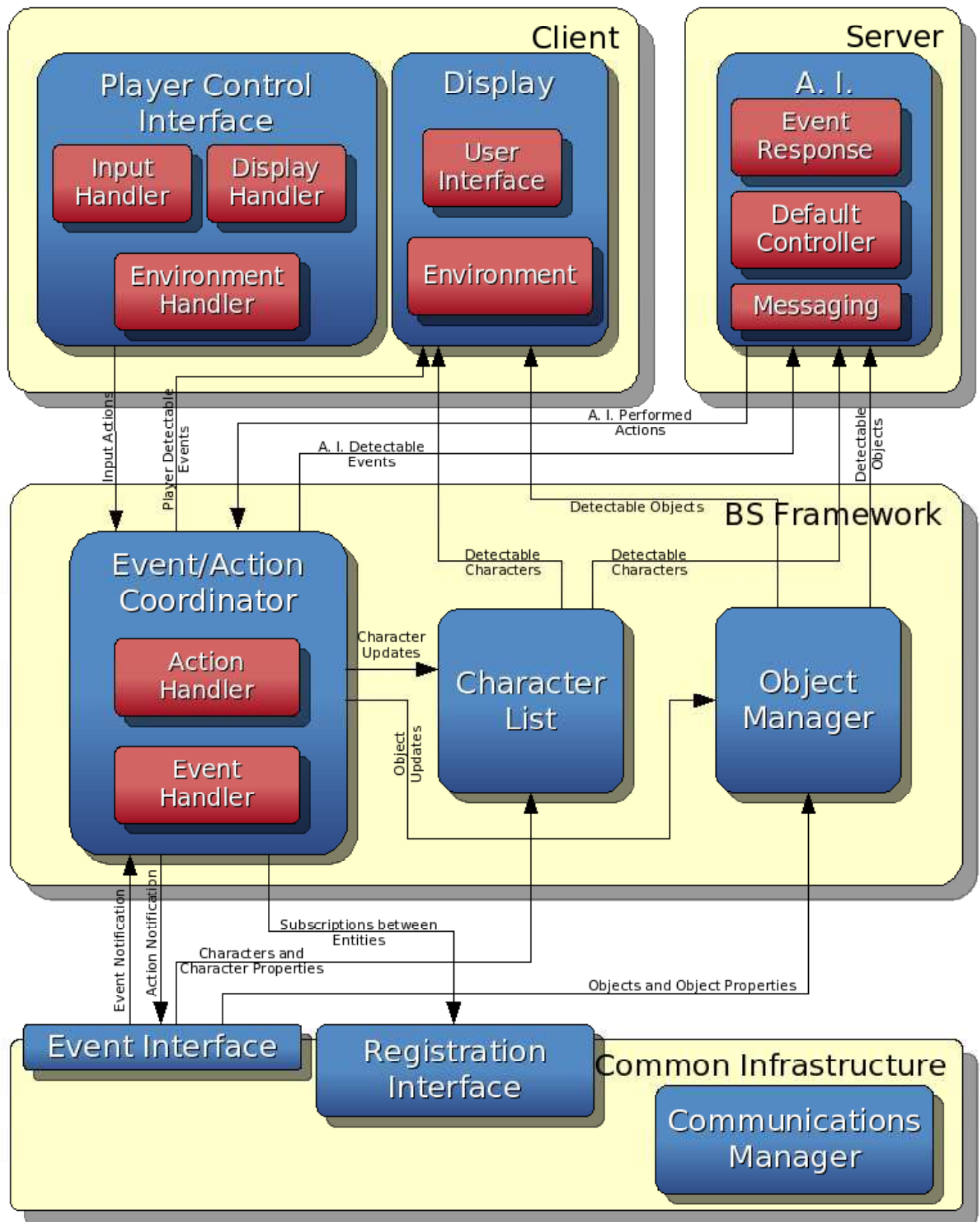
**Version 1.0**

### **Project Members**

- Matthew Fulgo
- Lane Haury
- Craig Szczesiul
- Tal Ron

### **Architectural Overview**

**See Attachments for Class Diagram**



## Player Control Interface

```
/* Fulfills Requirement(s):
   B.1.c, C.2.c.i

   This class is the main entry point for player inputs of all types.
   It thus addresses both requirements.
*/
public class InputHandler {
    public InputHandler();
    public void setDisplay(Display display);
    public DisplayHandler getDisplayHandler();
    public EnvironmentHandler getEnvironmentHandler();
}

/* Fulfills Requirement(s):
   B.1.d, B.1.f

   Both Communication and Environmental Events must be passed
   to the EAC, so there is no need for two objects here.
*/
public class DisplayHandler {
    public DisplayHandler(Display display);
    public void inputReceived(InputEvent ie);
    public Display getDisplay();
}

/* Fulfills Requirement(s):
   B.1.a
*/
public class EnvironmentHandler {
    public EnvironmentHandler(Eac eac);
    public void inputReceived(InputEvent ie);
    public void communicationReceived(CommunicationEvent ce);
    public Eac getEac();
}

/* Fulfills Requirement(s):
   B.1.e
*/
public class InputEvent {
    public enum InputType
    {
        mouse(0),
        keyboard(1)
    };
    public InputType getInputType();
    public char getKey();
    public int getButton();
    public Point2D getClickLocation();
    public Event getAction();
}
```

## Artificial Intelligence

```
/* Fulfills Requirement(s):
   A.1.a, A.1.b, C.2.c.ii

   AI are allowed in the simulation. Furthermore, they will be treated
   all other players by external entities.
*/
public class Ai {
    public Ai();
    public void setEac(Eac eac);
    public AiEventHandler getAiEventHandler();
    public AiDefaultController getAiDefaultController();
    public AiCommunication getAiCommunication();
}
```

```

/* Fulfills Requirement(s):
   A.1.c
*/
public class AiEventHandler implements EventListener {
    // implements public eventReceived(Event e);
    public AiEventHandler(Eac eac);
    public Eac getEac();
}

/* Fulfills Requirement(s):
   A.1.d
*/
public class AiDefaultController {
    public AiDefaultController(Eac eac);
    public Eac getEac();
}

/* Fulfills Requirement(s):
   A.1.e
*/
public class AiCommunication implements EventListener {
    // implements public eventReceived(Event e);
    public AiCommunication(Eac eac);
    public Eac getEac();
}

```

## Event/Action Coordinator

```

/* Fulfills Requirement(s):
   B.1.b, C.1.a, C.1.b, C.1.c, C.1.d, C.1.g

   The EAC handles all routing of events; therefore, all requirements
   dealing with event routing should map onto the EAC. Also, because
   the EAC is keeping track of simulation time, it should handle
   sending an already-scheduled event.
*/
public interface Eac
{
    public void performAction(Event performed);
    public void addEventSubscription(SimulationCharacter char, EventListener listener);
    public void scheduleEvent(long simulationTime, Event e);
}

/* Fulfills Requirement(s):
   C.1.d, C.1.g
*/
public interface EventListener
{
    public void eventReceived(Event ce);
}

/* Fulfills Requirement(s):
   C.1.d.i, C.1.h
*/
public class Event
{
    public enum PublicityType
    {
        Public(0),
        Private(1),
        Universal(2);
    }
    public PublicityType getPublicityType();
    public void setOriginator(SimulationObject originator);
    public SimulationObject getOriginator();
    public void setRadiusOfInfluence(int radius);
    public int getRadiusOfInfluence();
}

```

```

/* Fulfills Requirement(s):
   C.1.e
*/
public class CommunicationEvent extends Event
{
    public enum CommunicationType
    {
        Talk(0),
        Shout(1),
        Radio(2),
        Whisper(3);
    }
    public CommunicationType getType();
    public String getMessage();
    public void setMessage(String message);
}
/* Fulfills Requirement(s):
   C.1.g, C.1.k
*/
public class MovementEvent extends Event
{
    public Point3d getOrigin();
    public void setOrigin(Point3d origin);
    public Point3d getNewLocation();
    public void setNewLocation(Point3d newLocation);
    public int getVelocity();
    public void setVelocity();
}

/* Fulfills Requirement(s):
   C.1.f, C.1.i, C.1.l

   Initiating police and fire alarms and acquiring objects
   will be handled as interactions.
*/
public class InteractionEvent extends Event
{
    public SimulationObject getTarget();
    public void setTarget(SimulationObject target);
    public int getInteractionId();
    public void setInteractionId();
}

/* Fulfills Requirement(s):
   C.1.l
*/
public class StateChangeEvent extends Event
{
    public SimulationObject getNewState();
    public void setNewState();
    public SimulationObject getOriginalState();
    public void setOriginalState();
}

/* Fulfills Requirement(s):
   C.1.g, C.1.j
*/
public class SoundEvent extends Event
{
    public Point3d getLocation();
    public void setLocation(Point3d source);
    public int getMagnitude();
    public void setMagnitude(int magnitude);
}

```

## Character List

```
/* Fulfills Requirement(s):
   C.2.a
*/
public class CharacterList
{
    public void add(SimulationCharacter sc){}
    public SimulationCharacter getCharacter(String name){}
    public SimulationCharacter remove(String name){}
    public SimulationCharacter findCharacterByValue(SimulationCharacter sc){}

    public SimulationCharacter[] findNearbyCharacters(Point3D point, double radius){}
}

/* Fulfills Requirement(s):
   C.2.b, C.2.d, C.2.e

   All of these requirements relate to character state,
   which should be held by a single object.
*/
public class SimulationCharacter extends SimulationObject
{
    public void setStatus(int status) {} //Updates the status of the player
    public int getStatus() {}
    public void addPossession(SimulationObject p) {} //Adds a possession
    public void removePossession(SimulationObject p) {} //Removes the possession
    public SimulationObject[] getPossessions() {}
    public void setTeam(int team){}
    public int getTeam() {}
    public String getName() {}
}
```

## Object Manager

```
/* Fulfills Requirement(s):
   C.3.a
*/
public class ObjectManager //extends the Infrastructure Team's Entity
{
    public void add(SimulationObject so){}
    public SimulationObject getObject(String name){}
    public SimulationObject remove(String name){}
    public SimulationObject findObjectByValue(SimulationObject sc){}

    public SimulationObject[] findNearbyObjects(Point3D point, double radius){}
}

/* Fulfills Requirement(s):
   C.3.b, C.3.c
*/
public class SimulationObject
{
    public void setLocation(Point3D loc) {} //Updates the x,y,z coordinates
    public Point3D getLocation() {}
    public void setVelocity(Vector3D velocity){}
    public Vector3D getVelocity() {}
    public boolean isMovable() {}
}
```

## Map

Provided by the Common Infrastructure Team

## Networking

Provided by the Common Infrastructure Team

## Attachments

- [Design.svg](#) (2.2 MB) - added by fulgo on 04/02/07 02:22:29.
- [Design.png](#) (497.5 kB) - added by fulgo on 04/02/07 02:22:58.