A FLEXIBLE AND EXPANDABLE ARCHITECTURE FOR COMPUTER GAMES

by

Jeff Plummer

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

ARIZONA STATE UNIVERSITY

December 2004

A FLEXIBLE AND EXPANDABLE ARCHITECTURE

FOR COMPUTER GAMES

by

Jeff Plummer

has been approved

November 2004

APPROVED:

____, Chair

____

____

Supervisory Committee

ACCEPTED:

____

Department Chair

____

Dean, Division of Graduate Studies

ABSTRACT

Computer games have grown considerably in scale and complexity since their humble beginnings in the 1960s. Modern day computer games have reached incredible levels of realism, especially in areas like graphics, physical simulation, and artificial intelligence. However, despite significant advances in software engineering, the development of computer games generally does not employ state-of-the-art software engineering practices and tools.

This thesis describes an architecture for computer games as a System of Systems where the computer game itself is emergent. The proposed architecture follows a data centered framework where the independent components collaborate on a central data store. The architecture offers capabilities that are essential in overcoming challenges faced in building computer games that can enjoy modifiability, expandability, and maintainability traits. The architecture promotes component-based development (e.g., commercial off the shelf components) since the collaborating components have loose couplings, which in turn facilitates systematic design integration of System of Systems.

TABLE OF CONTENTS

Page

LIST OF FIGURES xxiii

CHAPTER

1 INTRODUCTION 1

1.1 Motivation 1

1.1.1 The current Approach and Its Shortcomings 1

1.1.2 The Migration to COTS 4

1.1.3 Not a Game Engine 5

1.2 High Level Objectives and Goals 6

1.2.1 Architectural Requirement: Support COTS-Based Development 7

1.2.2 Architectural Requirement: Better Knowledge Localization 7

1.2.3 Architectural Requirement: Flexibility / Modifiability 8

1.2.4 Architectural Requirement: Expandability / Maintainability 9

1.2.5 Performance and Other Quality Attributes are NOT requirements 9

1.3 Contributions 10

2 LITERATURE REVIEW 11

2.1 Current State of Game Development in Literature 11

2.2 The Latest Book Trends in Game Development 13

2.3 The First and Only Real Attempt at Game Architecture 14

CHAPTER Page

2.4 Software Architecture 15

3 THESIS METHODOLOGY 17

3.1 Analysis of Games as Software Systems 18

3.1.1 Selecting Games to Analyze 18

3.1.1.1 Existing Game Genres 19

3.1.1.2 Further Refinement – Isolate Important Properties 21

3.1.2 The Selected Games for Analysis 23

3.1.3 Analyzing the Games 27

3.1.3.1 Analyzing Starcraft™ Requirements with Use-Cases 27

3.1.3.2 Understanding the Sub-System Interaction 30

3.2 Identify Candidate Architectural Styles 32

3.2.1 Layered 32

3.2.2 Data-Centered 32

3.2.3 Independent Components 33

3.2.4 Data Flow 33

3.2.5 System of Systems 33

3.3 Architecture Design 34

3.3.1 Choosing a Topology 34

3.3.1.1 Layered Architectural Style 35

3.3.1.2 Data Flow Architectural Style 36

3.3.1.3 Data Centered Architectural Style 38

3.3.1.4 Independent Components Architectural Style 40

CHAPTER Page

3.3.1.5 System of Systems ... 42

3.3.2 Making the Topology Choice ... 43

3.3.3 Choosing a Style of Communication ... 45

3.3.3.1 Repository ... 45

3.3.3.2 Blackboard ... 46

3.3.3.3 Making the Communications Choice ... 47

3.3.4 Synchronicity ... 47

3.3.4.1 Synchrous at the Object Level ... 47

3.3.4.2 Batch Synchronization ... 48

3.3.4.3 Hybrid Synchronization ... 48

3.3.4.4 Making the Synchronicity Choice ... 48

3.4 The Idea – System of Systems Philosophy ... 49

4 THE PROPOSED ARCHITECTURE (and a Simple Design) ... 50

4.1 The Data-Centered System of Systems Topology ... 50

4.2 Architecture – System Communication ... 53

4.3 Architecture – Synchronization ... 54

4.4 Architecture – Distributed Synchronization ... 55

4.5 Architectural Features / Architectural Requirements ... 58

4.5.1 Support for COTS-Based Development ... 58

4.5.2 Better Knowledge Localization ... 58

4.5.3 System Flexibility / Modifiability ... 58

CHAPTER Page

4.5.4 System Expandability / Maintainability ... 59

4.6 A Simple Design ... 60

4.6.1 Potential Design: System Communication / Interaction ... 60

4.6.2 Potential Design Cont.: Attaching Systems at Compile Time ... 61

4.6.3 Potential Design Cont.: System Communication ... 63

4.6.4 Potential Design Cont.: Observer Pattern to Achieve Localization of Domain Knowledge ... 65

5 ARCHITECTURE VALIDATION ... 68

5.1 Taking the Reference Games to the Design Level ... 68

5.1.1 Applying the Design ... 68

5.1.2 Evaluating the results of applying the design ... 73

5.2 Developing a Prototype ... 74

5.2.1 Prototype High Level Design ... 74

5.2.1.1 Component Selection ... 74

5.2.1.2 The Object Data ... 76

5.2.2 Prototype Detailed Design ... 77

5.2.2.1 Component Interfaces ... 78

5.2.2.2 Domain-specific System – Object System Interactions ... 81

5.2.2.2.1 Connecting Domain System to the Object System ... 81

5.2.2.2.2 “Ticking” the Domain-specific System ... 82

5.2.3 Prototype Evaluation ... 83

CHAPTER Page

6 RESULTS 85

6.1 Summary 85

6.2 Conclusions – Meeting The Architectural Requirements 86

6.2.1 Support COTS-Based Development 87

6.2.2 Better Knowledge Localization 87

6.2.3 Flexibility / Modifiability 88

6.2.4 Expandability / Maintainability 88

6.2.5 The Performance Concern 89

6.3 Important Considerations 90

6.3.1 Design is Critical 90

6.3.2 Central Object Management System = VERY different 91

6.3.3 Think about the Data 92

6.4 Future Research 93

6.4.1 Can this Architecture Work for Massively Multiplayer Online Games ... 93

6.4.2 Design: Domain-specific Component Connection to the Object Management Component 93

6.4.3 Design: No More Interfaces to Access Object Data (If performance allows) 94

6.4.4 Architecture Inside the Components 94

6.4.5 What is messaging overhead for independent component style 94

6.4.6 The Architectural Tradeoff Analysis Method……………………………95

CHAPTER Page

Works Cited 96

APPENDIX Page

APPENDIX A - GAME ANALYSES 99

A - 1.1 Game Analysis 107

A - 1.1.1 Game Analysis - Use Case and Dynamic View 107

A - 1.1.1.1.1.1.1.1.1 Player 107

A - 1.1.1.1.1.1.1.1.2 System 107

A - 1.1.1.1.1.1.1.1.3 System (Ticked) 108

A - 1.1.1.2 Modules 109

A - 1.1.1.2.1 Game Data 110

A - 1.1.1.2.2 Game Logic 110

A - 1.1.1.2.3 Technology Modules 110

A - 1.1.1.2.3.1 AI 110

A - 1.1.1.2.3.2 Audio 110

A - 1.1.1.2.3.3 Graphics 110

A - 1.1.1.2.3.4 Network 110

A - 1.1.1.2.3.5 Physics 110

A - 1.1.1.2.3.6 User Interface 110

A - 1.1.1.3 Starcraft 111

A - 1.1.1.3.1 Use Cases 111

A - 1.1.1.3.1.1 Startup 112

A - 1.1.1.3.1.1.1.1.1 Select Multi-Player Game 112

A - 1.1.1.3.1.1.1.1.2 Select Single Player Game 112

A - 1.1.1.3.1.2 Options Menu 114

A - 1.1.1.3.1.2.1.1.1 End Mission 114

APPENDIX Page

A - 1.1.1.3.1.2.1.1.2 Get Help 115

A - 1.1.1.3.1.2.1.1.3 Get Mission Objective 115

A - 1.1.1.3.1.2.1.1.4 Load Game 115

A - 1.1.1.3.1.2.1.1.5 Modify Options 115

A - 1.1.1.3.1.2.1.1.6 Return To Game 115

A - 1.1.1.3.1.2.1.1.7 Save Game 116

A - 1.1.1.3.1.3 Play Starcraft 117

A - 1.1.1.3.1.3.1.1.1 Attack Unit 117

A - 1.1.1.3.1.3.1.1.2 Change Map Display Area 123

A - 1.1.1.3.1.3.1.1.3 Gather Resources 126

A - 1.1.1.3.1.3.1.1.4 Give unit an order 132

A - 1.1.1.3.1.3.1.1.5 Move to Location 137

A - 1.1.1.3.1.3.1.1.6 Research Technology 142

A - 1.1.1.3.1.3.1.1.7 Select Object 145

A - 1.1.1.3.1.3.1.1.8 Building construct Unit 150

A - 1.1.1.3.1.3.1.1.9 Give Building an order 150

A - 1.1.1.3.1.3.1.1.10 Hold Position 150

A - 1.1.1.3.1.3.1.1.11 Manipulate Object Resources 151

A - 1.1.1.3.1.3.1.1.12 Manipulate Player Resources 151

A - 1.1.1.3.1.3.1.1.13 Modify Doable Commands 151

A - 1.1.1.3.1.3.1.1.14 Patrol Location 151

A - 1.1.1.3.1.3.1.1.15 Stop Movement 151

A - 1.1.1.3.1.3.1.1.16 Unit Construct Building 151

APPENDIX Page

A - 1.1.1.3.1.4 Design: Tick Starcraft System 153

A - 1.1.1.3.1.4.1.1.1 Tick Starcraft Game System 154

A - 1.1.1.3.1.4.2 Tick AI System 155

A - 1.1.1.3.1.4.2.1.1 Tick AI System 155

A - 1.1.1.3.1.4.2.1.2 Navigate Map - Pathfinding........................ 157

A - 1.1.1.3.1.4.2.1.3 Attack 158

A - 1.1.1.3.1.4.2.1.4 Calculate AI State 158

A - 1.1.1.3.1.4.2.1.5 Calculate Next Movement 159

A - 1.1.1.3.1.4.2.1.6 Calculate unit action 159

A - 1.1.1.3.1.4.2.1.7 Execute Map Watcher................................ 159

A - 1.1.1.3.1.4.3 Tick Audio System 161

A - 1.1.1.3.1.4.3.1.1 Tick Audio System 161

A - 1.1.1.3.1.4.4 Tick Graphics System... 164

A - 1.1.1.3.1.4.4.1.1 :IGraphicsObjectSystem 164

A - 1.1.1.3.1.4.4.1.2 Update View Object 164

A - 1.1.1.3.1.4.4.1.3 Tick Graphics System................................ 166

A - 1.1.1.3.1.4.4.1.4 Update View 166

A - 1.1.1.3.1.4.4.1.5 Update Main View..................................... 171

A - 1.1.1.3.1.4.4.1.6 Draw Main View Objects 171

A - 1.1.1.3.1.4.4.1.7 Draw Main View Terrain........................... 171

A - 1.1.1.3.1.4.4.1.8 Update All Views 171

A - 1.1.1.3.1.4.4.1.9 Update Command Button View.................. 171

A - 1.1.1.3.1.4.4.1.10 Update Mini Map View 172

A - 1.1.1.3.1.4.4.1.11 Update Protrait View 172

A - 1.1.1.3.1.4.4.1.12 Update Status View 172

A - 1.1.1.3.1.4.5 Tick Network Component 173

A - 1.1.1.3.1.4.5.1.1 Broadcast local objects TO server 173

A - 1.1.1.3.1.4.5.1.2 Tick Network System 173

A - 1.1.1.3.1.4.5.1.3 Update objects FROM server 175

A - 1.1.1.3.1.4.6 Tick Object Component... 176

A - 1.1.1.3.1.4.6.1.1 Tick Object System / Game Logic............. 176

A - 1.1.1.3.1.4.6.1.2 Update Commander Object 178

A - 1.1.1.3.1.4.6.1.3 Update Controlled Object 178

A - 1.1.1.3.1.4.7 Tick UI Component 179

A - 1.1.1.3.1.4.7.1.1 Process Keyboard 179

A - 1.1.1.3.1.4.7.1.2 Process Mouse 179

A - 1.1.1.3.1.4.7.1.3 Tick User Interface 179

A - 1.1.1.4 Unreal Tournament.. 182

A - 1.1.1.4.1 Use Cases.. 182

A - 1.1.1.4.1.1 Play Unreal Tournament.. 183

A - 1.1.1.4.1.1.1.1.1 Collect Ammo.. 183

A - 1.1.1.4.1.1.1.1.2 Collect Health 183

A - 1.1.1.4.1.1.1.1.3 Collect Item 183

A - 1.1.1.4.1.1.1.1.4 Collect Weapon 185

A - 1.1.1.4.1.1.1.1.5 Jump.. 185

A - 1.1.1.4.1.1.1.1.6 Move 185

A - 1.1.1.4.1.1.1.1.7 Rotate 187

A - 1.1.1.4.1.1.1.1.8 Shoot 187

A - 1.1.1.4.1.2 Design: Tick.. 188

A - 1.1.1.4.1.2.1.1.1 System (Ticked)... 188

A - 1.1.1.4.1.2.1.1.2 Tick Physics Component 189

A - 1.1.1.4.1.2.1.1.3 Tick AI System 189

A - 1.1.1.4.1.2.1.1.4 Tick Audio Component 189

A - 1.1.1.4.1.2.1.1.5 Tick Graphics 3D Component 189

A - 1.1.1.4.1.2.1.1.6 Note... 189

A - 1.1.1.4.1.2.1.1.7 Tick Network Component 190

A - 1.1.1.4.1.2.1.1.8 Tick Unreal Tournament Game System 190

A - 1.1.1.4.1.2.2 Tick AI System 191

A - 1.1.1.4.1.2.2.1.1 Tick Unreal Tournament Game System 191

A - 1.1.1.4.1.2.2.1.2 System (Ticked)... 191

A - 1.1.1.4.1.2.2.1.3 Note... 191

A - 1.1.1.4.1.2.2.1.4 Tick AI System 191

A - 1.1.1.4.1.2.2.1.5 Tick Player.. 193

A - 1.1.1.4.1.2.2.1.6 Tick Projectile.. 193

A - 1.1.1.4.1.2.3 Tick Audio Component 194

A - 1.1.1.4.1.2.3.1.1 Tick Audio Component 194

A - 1.1.1.4.1.2.4 Tick Graphics 3D Component................................. 196

A - 1.1.1.4.1.2.4.1.1 Tick Graphics 3D Component 196

A - 1.1.1.4.1.2.4.1.2 Update All Graphical Views....................... 198

APPENDIX Page

A - 1.1.1.4.1.2.4.1.3 Update Character Status Overlay 198

A - 1.1.1.4.1.2.4.1.4 Update GUI Overlays 198

A - 1.1.1.4.1.2.4.1.5 Update Main Play View 198

A - 1.1.1.4.1.2.4.1.6 Update Team Score Overlay 200

A - 1.1.1.4.1.2.4.1.7 Update Weapon/Ammo Overlay 200

A - 1.1.1.4.1.2.5 Tick Network Component 201

A - 1.1.1.4.1.2.5.1.1 Broadcast Local Objects TO Server 201

A - 1.1.1.4.1.2.5.1.2 Tick Network Component 201

A - 1.1.1.4.1.2.5.1.3 Update Local Objects FROM Server 203

A - 1.1.1.4.1.2.6 Tick Object Component 204

A - 1.1.1.4.1.2.6.1.1 Tick Object Component 204

A - 1.1.1.4.1.2.7 Tick Physics Component 207

A - 1.1.1.4.1.2.7.1.1 Calculate Collision Reaction 207

A - 1.1.1.4.1.2.7.1.2 Detect Collisions 207

A - 1.1.1.4.1.2.7.1.3 Tick Physics Component 207

APPENDIX B – PROTOTYPE DESIGN 210

B - 1.2 Prototype 218

B - 1.2.1 Analysis View 218

B - 1.2.1.1 Logical Architecture 218

B - 1.2.1.1.1 Object Interfaces 219

B - 1.2.1.1.1.1.1.1.1 GameObject 219

B - 1.2.1.1.1.1.1.1.2 AI2Object 220

B - 1.2.1.1.1.1.1.1.3 IAIObject 220

APPENDIX Page

B - 1.2.1.1.1.1.1.1.4 IGraphics2DObject 220

B - 1.2.1.1.1.1.1.1.5 IGraphics3DObject 221

B - 1.2.2 Logical View... 222

B - 1.2.2.1 Programming Utilities Library.. 222

B - 1.2.2.2 Systems 223

B - 1.2.2.1.2 AI System 224

B - 1.2.2.1.1 AI Component - Implementation 224

B - 1.2.2.2.1.1.1 AI Exported Classes.. 225

B - 1.2.2.2.1.1.1.1.1 Root.. 225

B - 1.2.2.2.1.1.2 Private AI System Implementation........................... 227

B - 1.2.2.2.1.1.2.1.1 CAISystem.. 227

B - 1.2.2.2.1.1.2.1.2 CAIProcessorObject 228

B - 1.2.2.2.1.1.2.1.3 CAIViewProcessor 229

B - 1.2.2.1.2 AI Component - Interfaces.. 231

B - 1.2.2.2.1.2.1 AI Interfaces Object System Can Use To Communicate With AI System 232

B - 1.2.2.2.1.2.1.1.1 IAIProcessorObject.................................... 232

B - 1.2.2.2.1.2.1.1.2 IAISystem 232

B - 1.2.2.2.1.2.1.1.3 IAIViewProcessor...................................... 233

B - 1.2.2.2.1.2.2 AI Interfaces The Object System Implements.......... 234

B - 1.2.2.2.1.2.2.1.1 IAICapableObject 234

B - 1.2.2.2.1.2.2.1.2 IAIObjectSystem.. 234

B - 1.2.2.2.1.2.2.1.3 IAIProcessableObject................................. 235

B - 1.2.2.2.1.2.2.1.4 IAISceneManager 236
B - 1.2.2.2.1.2.2.1.5 IAIView 236
B - 1.2.2.2.2 AI2System 238
B - 1.2.2.2.2.1 AI2 Component - Implementation 238
B - 1.2.2.2.2.1.1 AI2 Exported Classes.. 239
B - 1.2.2.2.2.1.1.1.1 Root.. 239
B - 1.2.2.2.2.1.2 Private AI2 System Implementation......................... 241
B - 1.2.2.2.2.1.2.1.1 CAI2System.. 241
B - 1.2.2.2.2.1.2.1.2 CAI2ProcessorObject 242
B - 1.2.2.2.2.1.2.1.3 CAI2ViewProcessor 243
B - 1.2.2.2.2 AI2 Component - Interfaces... 245
B - 1.2.2.2.2.2.1 AI2 Interfaces Object System Can Use To Communicate With AI2 System... 246
B - 1.2.2.2.2.2.1.1.1 IAI2ProcessorObject.................................. 246
B - 1.2.2.2.2.2.1.1.2 IAI2System 246
B - 1.2.2.2.2.2.1.1.3 IAI2ViewProcessor.................................... 247
B - 1.2.2.2.2.2.2 AI2 Interfaces The Object System Implements........ 248
B - 1.2.2.2.2.2.2.1.1 IAI2CapableObject 248
B - 1.2.2.2.2.2.2.1.2 IAI2ObjectSystem..................................... 248
B - 1.2.2.2.2.2.2.1.3 IAI2ProcessableObject 249
B - 1.2.2.2.2.2.2.1.4 IAI2SceneManager 250
B - 1.2.2.2.2.2.2.1.5 IAI2View 250
B - 1.2.2.3.2 Game Object System 252

APPENDIX Page

B - 1.2.2.3.1 Game Object Component - Implementation 252

B - 1.2.2.2.3.1.1 Game Object Component Exported Classes 252

B - 1.2.2.2.3.1.1.1.1 Root 252

B - 1.2.2.2.3.1.2 Private Game Object Component Implementation ... 254

B - 1.2.2.2.3.1.2.1.1 CDemoCamera 254

B - 1.2.2.2.3.1.2.1.2 CDemoGameObjectSystem 255

B - 1.2.2.2.3.1.2.1.3 CDemoMainView 259

B - 1.2.2.2.3.1.2.1.4 CDemoObject 259

B - 1.2.2.2.3.1.2.1.5 CDemoObjectSceneManager 265

B - 1.2.2.2.3.1.2.1.6 CDemoViewBaseClass 267

B - 1.2.2.2.3.1.2.1.7 CTriangleGameObject 273

B - 1.2.2.2.3.1.2.2 Data Structures 275

B - 1.2.2.2.3.1.2.2.1 demoPoint2i 275

B - 1.2.2.2.3.1.2.2.2 demoPoint3f 276

B - 1.2.2.2.3.1.2.2.3 demoRect 276

B - 1.2.2.3.2 Game Object Component - Interfaces 278

B - 1.2.2.2.3.2.1.1.1 IObjectSystem 278

B - 1.2.2.3.3 Component Attachings 279

B - 1.2.2.4.2 Game System 281

B - 1.2.2.2.4.1.1.1.1 CDemoApplication 281

B - 1.2.2.5.2 Graphic 3D System 284

B - 1.2.2.5.1 Graphics3DComponent - Implementation 284

B - 1.2.2.2.5.1.1 Exported Classes 285

B - 1.2.2.2.5.1.1.1.1 Root 285
B - 1.2.2.2.5.1.2 Private Graphics3D System Implementation 287
B - 1.2.2.2.5.1.2.1.1 CGraphics3DProcessorObject 287
B - 1.2.2.2.5.1.2.1.2 CGraphics3DSystem 288
B - 1.2.2.2.5.1.2.1.3 CGraphics3DViewProcessor 291
B - 1.2.2.5.2 Graphics3DComponent - Interfaces 293
B - 1.2.2.2.5.2.1 Interfaces the Object System can use to communicate with the Graphics3D System 294
B - 1.2.2.2.5.2.1.1.1 IGraphics3DProcessorObject 294
B - 1.2.2.2.5.2.1.1.2 IGraphics3DSystem 295
B - 1.2.2.2.5.2.1.1.3 IGraphics3DViewProcessor 295
B - 1.2.2.2.5.2.2 Interfaces The Object System Implements 297
B - 1.2.2.2.5.2.2.1.1 IGraphics3DCamera 297
B - 1.2.2.2.5.2.2.1.2 IGraphics3DCapableObject 297
B - 1.2.2.2.5.2.2.1.3 IGraphics3DObjectSystem 298
B - 1.2.2.2.5.2.2.1.4 IGraphics3DProcessableObject 298
B - 1.2.2.2.5.2.2.1.5 IGraphics3DSceneManager 299
B - 1.2.2.2.5.2.2.1.6 IGraphics3DView 300
B - 1.2.2.6.2 Graphics 2D System 302
B - 1.2.2.6.1 Graphics Component - Implementation 302
B - 1.2.2.2.6.1.1 Exported Classes 303
B - 1.2.2.2.6.1.1.1.1 Root 303
B - 1.2.2.2.6.1.2 Private Graphics System Implementation 305

APPENDIX Page

B - 1.2.2.2.6.1.2.1.1 CGraphicsProcessorObject 305

B - 1.2.2.2.6.1.2.1.2 CGraphicsSystem...................................... 308

B - 1.2.2.2.6.1.2.1.3 CGraphicsViewProcessor 310

B - 1.2.2.6.2 Graphics Component - Interfaces 312

B - 1.2.2.2.6.2.1 Interfaces Object System Can Use To Communicate With Graphics System 313

B - 1.2.2.2.6.2.1.1.1 IGraphicsProcessorObject.......................... 313

B - 1.2.2.2.6.2.1.1.2 IGraphicsSystem....................................... 313

B - 1.2.2.2.6.2.2 Interfaces The Object System Implements 315

B - 1.2.2.2.6.2.2.1.1 I2DGraphicsCamera 315

B - 1.2.2.2.6.2.2.1.2 I2DGraphicsObject 316

B - 1.2.2.2.6.2.2.1.3 I2DSpriteGraphicsObject........................... 316

B - 1.2.2.2.6.2.2.1.4 IGraphicsCamera 317

B - 1.2.2.2.6.2.2.1.5 IGraphicsCapableObject............................ 317

B - 1.2.2.2.6.2.2.1.6 IGraphicsObjectIterator 317

B - 1.2.2.2.6.2.2.1.7 IGraphicsObjectSystem 318

B - 1.2.2.2.6.2.2.1.8 IGraphicsSceneManager............................ 318

B - 1.2.2.2.6.2.2.1.9 IGraphicsView 319

B - 1.2.2.2.6.2.2.1.10 IGraphicsViewIterator 320

B - 1.2.2.2.6.2.2.1.11 IProcessableGraphicsObject 321

B - 1.2.2.3 Utility Includes.. 323

B - 1.2.2.3.1.1.1.1.1 CStdStr.. 323

B - 1.2.2.3.1.1.1.1.2 IIterator 334

B - 1.2.2.3.1.1.1.1.3 VectorBasedIteratorTemplateClass 335

B - 1.2.3 Dynamic View 337

B - 1.2.3.1 Initialize 337

B - 1.2.3.1.1.1.1.1.1 Initialize AI2 System 337

B - 1.2.3.1.1.1.1.1.2 Initialize AI System 340

B - 1.2.3.1.1.1.1.1.3 Initialize Graphics 3D System 343

B - 1.2.3.1.1.1.1.1.4 Initialize Graphics System 346

B - 1.2.3.1.1.1.1.1.5 Initialize Object System 350

B - 1.2.3.1.1.1.1.1.6 Initialize Game System 353

B - 1.2.3.2 Tick 356

B - 1.2.3.2.1.1.1.1.1 Tick AI System 356

B - 1.2.3.2.1.1.1.1.2 Tick AI2 System 362

B - 1.2.3.2.1.1.1.1.3 Tick Graphics 3D System 367

B - 1.2.3.2.1.1.1.1.4 Tick Graphics System 373

B - 1.2.3.2.1.1.1.1.5 Tick Prototype Game System 379

B - 1.2.4 Component View 380

B - 1.2.4.1.1.1.1.1.1 AI System 2... 380

B - 1.2.4.1.1.1.1.1.2 Artificial Intelligence 380

B - 1.2.4.1.1.1.1.1.3 Audio... 380

B - 1.2.4.1.1.1.1.1.4 Game System 380

B - 1.2.4.1.1.1.1.1.5 Graphics 381

B - 1.2.4.1.1.1.1.1.6 Graphics 3D System 381

B - 1.2.4.1.1.1.1.1.7 Network.. 381

APPENDIX Page

B - 1.2.4.1.1.1.1.1.8 Object & Object Management System (Data) 381

B - 1.2.4.1.1.1.1.1.9 OGRE Graphics Engine 381

B - 1.2.4.1.1.1.1.1.10 Physics Component 382

B - 1.2.4.1.1.1.1.1.11 User Interface 382

LIST OF FIGURES

Figure Page

1 - Rollings' and Morris' Game Architecture *2*

2 - Object Centric View of Games *4*

3 - Current Object Centered COTS Approach *5*

4 - Object/Class Level Separation of Logic *12*

5 Rollings' and Morris' Game Architecture *15*

6 - Screenshot from the Game Starcraft *24*

7 - Screenshot from Unreal Tournament *26*

8 - Screenshot Unreal Tournament 2004 ..... *26*

9 - Playing Starcraft Use Case Diagram *28*

10 - Logical Modules *29*

11 - Select Object (Subsystem interactions) *31*

12- A Simple Layered Architecture *35*

13- Data Flow *37*

14- Data Flow at the Component Level (AI) *38*

15 – Data Centered *39*

16 – Select Object (Logical Module Interactions – Data Centered) *40*

17- Independent Components *42*

18 - Layered and Data-Centered *45*

19 - Repository *46*

20 - Data Centered System of Systems *51*

Figure Page

21- Intelligent Data System Centered System of Systems *52*

22 – System Defined as a Domain-specific Component & the Object Component *53*

23 - Ticking the Game System of Systems *55*

24 – Example Peer to Peer Networked Game *56*

25 -Example Client Server Networked Game *57*

26- Potential Design using many AI Systems *59*

27 – Interfaces Required to Connect Domain-specific Component to the Object Management Component *62*

28 – Example Sequence of Connecting a Domain-specific Component to the Object Management Component *62*

29 – Interfaces Required for Domain-specific System To Request Objects to Process *64*

30 – Example Sequence of a Domain-specific System Requesting Objects to Process *65*

31-Potential Design using a Domain Observer Object *66*

32-Potential Sequence using a Domain Observer Object *67*

33 - Tick Game System Use Case *70*

34 – Tick Graphics System *71*

35 – Update View Component Sequence *72*

36 – Update View – Classes and Interfaces *73*

37 – Prototype Subsystems *75*

38 – Analysis of Object Data Required *77*

39 - Example: Graphics3D System Interfaces *79*

40 – Interfaces Into the Graphics 3D System *80*

Figure Page

41 – Interfaces the Object and Object Management System Must Implement in order for the Graphics 3D Component to Use it. *81*

42 – Connecting the Object Component to the Graphics3D Component *82*

43 – Prototype Sequence: Tick Graphics2D System *83*

44 – Screenshot1 from Prototype *84*

45 - Screenshot 2 from Prototype *84*

46 : Analysis *107*

47 : Logical Modules *109*

48 : Use Case Model *111*

49 : Startup *112*

50 : Options Menu *114*

51 : Play Starcraft *117*

52 : Analysis: Attack Unit (Logical Modules Involved) *118*

53 : Design: Attack Unit (Component Sequence) *121*

54 : Analysis: Change Map Display Area by Moving Mouse to Edge of Screen(Logical Modules Involved) *124*

55 : Design: Change Map Display Area (Component Sequence) *125*

56 : Analysis: Gather Resources (Logical Modules Involved) *127*

57 : Design: Gather Resources (Component Sequence) *130*

58 : Analysis: Give unit an order by clicking order button (Logical Modules Involved)133

59 : Design: Give unit an order (Component Sequence) *135*

60 : Analysis: Move to Location (Sub-system Interactions) *137*

61 : Design: Move to Location (Component Sequence) *140*

Figure Page

62 : Analysis: Research Technology (Sub-System Interaction) *143*

63 : Design: Research Technology (Component Sequence) *144*

64 : Analysis: Select Object (Logical Modules Involved) *146*

65 : Design: Select Object (Component Sequence) *148*

66 : Tick Starcraft Game System *153*

67 : Tick AI System *155*

68 : Design: Tick AI System (Component Sequence) *156*

69 : Design: Navigate Map - Pathfinding (Component Sequence) *157*

70 : Tick Audio System *161*

71 : Design: Tick Audio System (Component Sequence) *162*

72 : Tick Graphics Component *164*

73 : Design: Update View Object (Component Sequence) *165*

74 : Design: Update View - (Component Sequence) *167*

75 : Design: Update View (Class-Interface Sequence) *168*

76 : Tick Network Component *173*

77 : Design: Tick Network System (Component Sequence) *174*

78 : Tick Object Component *176*

79 : Design: Tick Object / Game Logic System (Component Sequence) *177*

80 : Tick UI Component *179*

81 : Design: Tick User Interface (Component Sequence) *180*

82 : Use Cases Model *182*

83 : Play Unreal Tournament *183*

84 : Analysis: Collect Item (Logical Modules Involved) *184*

Figure Page

85 : Analysis: Move (Logical Modules Involved) *186*

86 : Design: Tick *188*

87 : Tick AI System *191*

88 : Design: Tick AI System (Component Sequence) *192*

89 : Tick Audio Component *194*

90 : Design: Tick Audio System (Component Sequence) *195*

91 : Tick Graphics 3D Component *196*

92 : Design: Tick Graphics 3D Component (Component Sequence) *197*

93 : Design: Update Main Play View (Component Sequence) *199*

94 : Tick Network Component *201*

95 : Design: Tick Network System (Component Sequence) *202*

96 : Tick Object Component *204*

97 : Design: Tick Object Component(Component Sequence) *205*

98 : Tick Physics Component *207*

99 : Design: Tick Physics Component (Component Sequence) *208*

100 : Prototype Logical Architecture *218*

101 : Required Object Interfaces *219*

102 : Programming Utilities Library *222*

103 : Systems *223*

104 : AI Component - Example Implementation *224*

105 : Exported Classes *225*

106 : Private AI System Implementation *227*

107 : AI Component - Public Interfaces *231*

Figure Page

108 : Interfaces Object System Can Use To Communicate With AI System................... 232

109 : Interfaces The Object System Implements... 234

110 : AI2 Component - Example Implementation... 238

111 : AI2 Exported Classes.. 239

112 : Private AI2 System Implementation 241

113 : AI2 Component - Interfaces.. 245

114 : AI2 Interfaces Object System Can Use To Communicate With AI2 System.......... 246

115 : AI2 Interfaces The Object System Implements 248

116 : Game Object Component Exported Classes.. 252

117 : Private Game Object Component Implementation... 254

118 : Game Object System - Data Structures 275

119 : Game Object Component - Interfaces 278

120 : Game Object System - AI Interface Implementations... 279

121 : Game Object System - AI2 Interface Implementations... 279

122 : Game Object System - Graphic Interface Implementations................................. 280

123 : Game Object System - Graphic3D Interface Implementations 280

124 : Game System.. 281

125 : Graphics3DComponent - Implementation.. 284

126 : Exported Classes 285

127 : Private Graphics3D System Implementation 287

128 : Graphics3DComponent - Interfaces... 293

129 : Interfaces the Object System can use to communicate with the Graphics3D System 294

Figure Page

130 : Interfaces The Object System Implements *297*

131 : Graphics Component - Implementation *302*

132 : Exported Classes *303*

133 : Private Graphics System Implementation *305*

134 : Graphics Component - Interfaces *312*

135 : Interfaces The Graphics System Implements *313*

136 : Interfaces The Object System Must Implement *315*

137 : Utility Includes *323*

138 : Initialize *337*

139 : Design: Initialize AI2 System (Component Sequence) *338*

140 : Design: Initialize AI2 System (Class-Interface Sequence) *339*

141 : Design: Initialize AI System (Component Sequence) *341*

142 : Design: Initialize AI System (Class-Interface Sequence) *342*

143 : Design: Initialize Graphics 3D System (Component Sequence) *343*

144 : Design: Initialize Graphics 3D System (Class-Interface Sequence) *344*

145 : Design: Initialize Graphics System - (Component Sequence) *347*

146 : Design: Initialize Graphics System (Class-Interface Sequence) *348*

147 : Design: Initialize Object System - (Component Sequence) *351*

148 : Design: Initialize Object System (Class-Interface Sequence) *352*

149 : Design: Initialize Game System - (Component Sequence) *353*

150 : Tick *356*

151 : Design: Tick AI System - (Component Sequence) *357*

152 : Design: Tick AI System (Class-Interface Sequence) *359*

Figure Page

153 : Design: Tick AI2 System (Component Sequence).. *362*

154 : Design: Tick AI2 System (Class-Interface Sequence) *364*

155 : Design: Tick Graphics3DSystem (Component Sequence)..................................... *367*

156 : Design: Tick Graphics3D System (Class-Interface Sequence) *369*

157 : Design: Tick Graphics System (Component Sequence) *374*

158 : Design: Tick Graphics System (Class-Interface Sequence) *375*

159 : Prototype Component Model.. *380*

1 INTRODUCTION

1.1 Motivation

Electronic games are a billion dollar industry developing software system commonly reaching into the millions of lines of code (“3 Million”), but the development process remains very much unchanged from the early days of programming (“A $30 Billion Industry”). It’s not unusual for development houses to move from the game idea directly to coding, where the success or failure depends almost entirely on the skill and experience level of the developers (Rollings 164-165). A base architecture that unifies the interaction between game subsystems and still allows for flexibility and expandability could greatly impact development the electronic entertainment industry.

1.1.1 The current Approach and Its Shortcomings

The current approach is to design and develop a custom architecture for each game. A game development house may carry over portions of a design from one game to another, but this is usually the result of individual experience rather than a formal design approach. So while skilled developers are still able to achieve the desired results, it is rarely on time and on schedule (Fristrom).

One problem with such an ad hoc approach to creating a game architecture is that quality attributes like flexibility and expandability are rarely incorporated in the design. For example ID™ software ended up rewriting almost all the code when moving from the game Quake™ 3 to Doom™ 3 (Sloan). Both are first person shooters, with the same game play. In fact the only noticeable difference is improved graphics. Since the game is primarily a graphical improvement, then the obvious culprit is the existing architecture didn’t lend itself to expandability. ID’s™ experience is definitely not unique. Countless companies waste time rewriting music code, GUI code, etc. simply because the existing code doesn't fit into the new game.

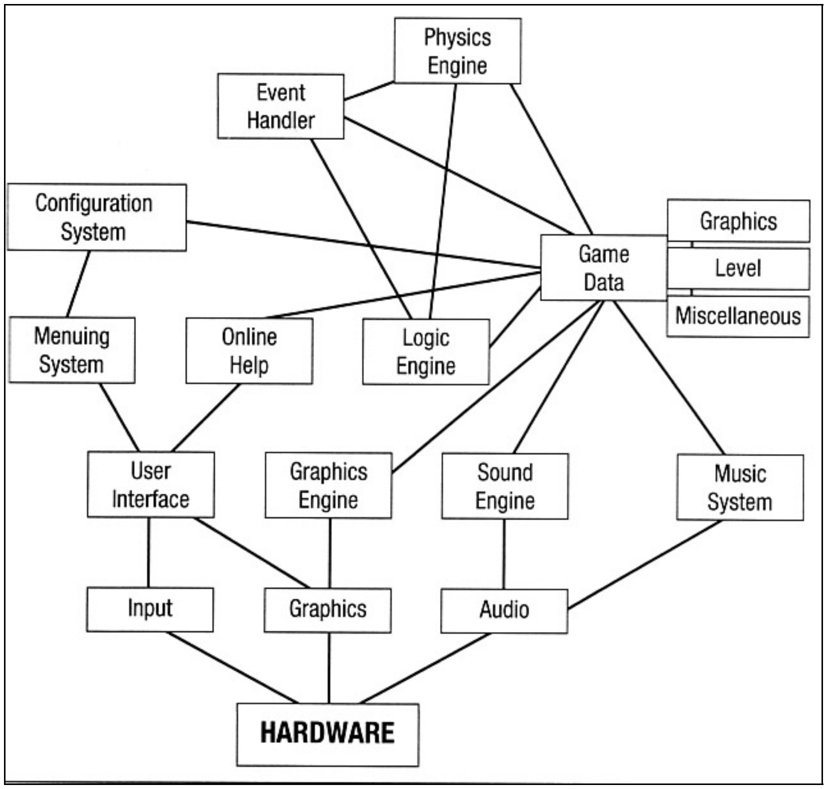


Figure 1 - Rollings' and Morris' Game Architecture

Rollings and Morris, the authors of *Game Architecture and Design*, reviewed existing game architectures, and attempted to map out a possible separation of logic (see Figure 1 above). While the component layout from Figure 1 may work for a game, I would argue the webbing of interrelated dependencies among subsystems would greatly limit the amount of expandability and re-use between game projects. A suitable architecture should not only have a logical separation of sub-systems, but also allow for any of those sub-systems to be easily swapped out or modified without breaking the overall system.

Part of the reason most attempts at a game architecture have a great deal of interdependencies is because of underlying object-centric view of games (see Figure 2 below). Games have always been about game objects living in a virtual world. Game objects have their own behavior, draw themselves to the screen, and even make their own sounds. This view makes sense logically, and seems to follow the widely accepted object-oriented paradigm. This view, however, is starting to show its limitations as the complexity for such functionality as drawing and thinking continue to climb exponentially. Such complexity has made game objects unwieldy and difficult to design around.

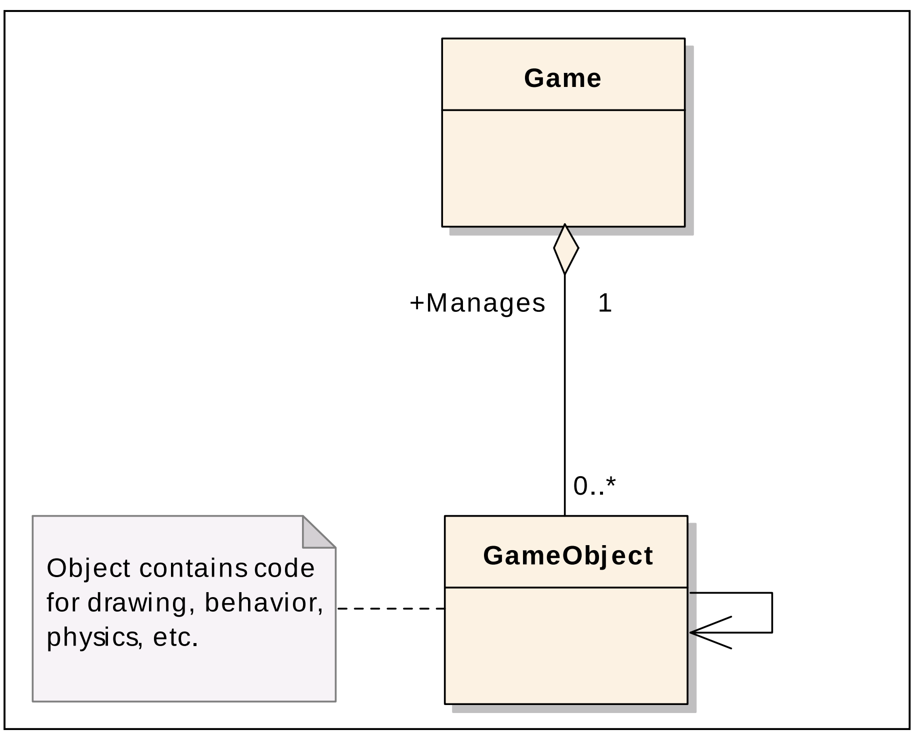


Figure 2 - Object Centric View of Games

1.1.2 The Migration to COTS

Software practices in games are undergoing a massive revolution. Games are approaching the production value of blockbuster movies, but without the same level of modularity and outsourcing. Movies are created by a several individual companies each specialized in areas like sound, special effects, etc. This level of separation of labor results in outstanding quality, and the ability to plan a timeline down to the actual camera shot. Games are just beginning this transition from 100% in-house code, to more of a Component Off the Shelf (COTS) based approach (Adolph).

Migration to COTS based systems is the first step in improving games on a massive scale. Allowing companies to focus on a single specialty means software technology can advance at a faster rate, and those advances are available for more games to use. While using COTS components can improve quality and time to develop (Alves et al. 1), staying with the current object-centric view means components are rarely more than functional libraries designed to help the object operate. Game objects are still responsible for all their own data manipulation including: graphics drawing, artificial intelligence (AI), sound, physics, etc. While games will still benefit from the technology COTS offers, it still means game objects are extremely large and complex. It also means game object developers must have a very strong knowledge about all the COTS components they are using to implement that object (See Figure 3 below).

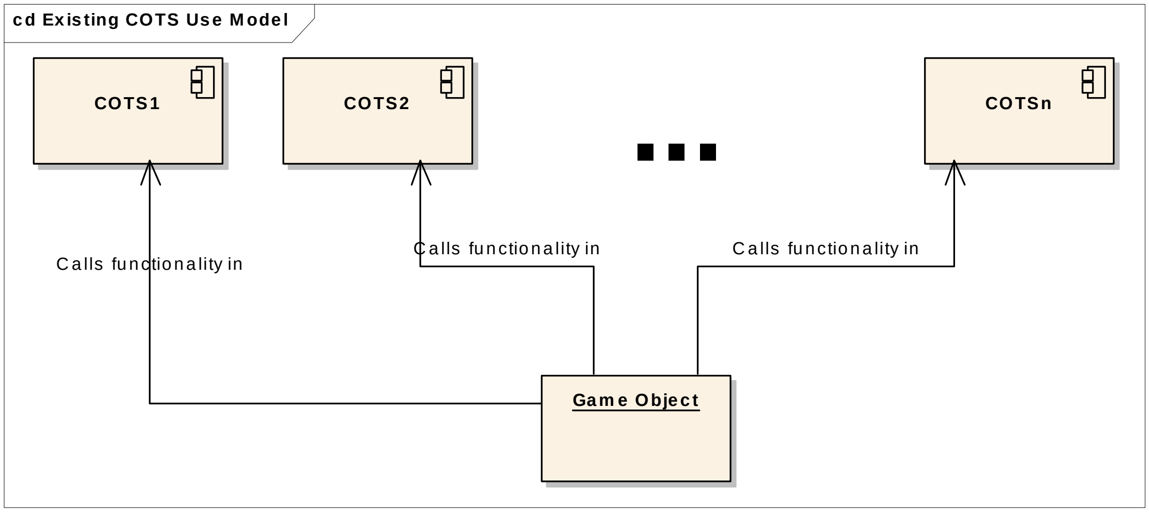


Figure 3 - Current Object Centered COTS Approach

The object-centric view also limits re-use, even when using COTS components. The object code is the least re-usable when moving between game projects, but it is the object code that contains the calls to the various components. So when moving from one project to another, developers will often have to re-write those same interactions. While a well-designed class hierarchy can mitigate much of that risk, the objects are still strongly tied to the COTS components they use. I propose there exists an architecture that can further increase code-reuse while reducing coupling.

1.1.3 Not a Game Engine

A common trend emerging in the game industry is the introduction to the all-encompassing COTS "game engine". Development houses can purchase very powerful "game engines", allowing the developers to develop a game using a commercially proven game framework. While this approach is an outstanding example of code re-use, it can limit the flexibility of the developers to design the game of their choosing. "Game engines" can limit the developers in a variety of ways.

- Limits due to engine design – "Game engines" were built with an initial game in mind, and the completed design reflects this intent. Trying to use the UnrealEngine™, the game engine used to create the first person shooter game Unreal™, to create a console style football game may prove to be a very laborious task.
- Limits due to cost – "Game engines" can be very expensive. Top-tier game engines can cost in the hundreds of thousands of dollars ("3D Engines"). The decision to use such an engine means the game developed must be mass marketable in order to recoup that large initial investment. Unfortunately, in order to have mass market appeal the developer has significantly reduced options in what kind of game to create.

The intent of this thesis is to design at a higher level of abstraction than the design of game engine. This is not to say a reusable commercial game engine could not be developed using the proposed architecture, but the distinction should be made between an architecture and a fully fleshed out system design.

1.2 High Level Objectives and Goals

The main objective of this thesis is to design and prove there exists a software architecture that is both expandable enough to grow with the technology and flexible enough to support the diverse world of games. Such an architecture would provide a starting place for game developers to begin from, and perhaps the start of a standardized communication between components used in a game system. A successful architecture will scale with the complexities of today's games, without sacrificing the developer's creative control over the game project. To achieve this rather lofty goal, the resulting architecture must fulfill the following requirements:

1.2.1 Architectural Requirement: Support COTS-Based Development

First the architecture must have strong separation of logic. The idea is to more completely separate the logic such that game subsystems can be independently developed and tested. This requirement is consistent with COTS based systems, and this thesis intends to continue with the COTS based approach.

In order to verify the resulting architecture meets this requirement it must be demonstratable that components can be independantly developed and tested. These components should be easy to integrate into a game application without a great deal of re-write on the part of the game. Ideally components will integrate in a similar yet logical fashion.

1.2.2 Architectural Requirement: Better Knowledge Localization

The architectural requirement of better knowledge localization exists because of the diverse capabilities required in games. Modern day games require outstanding graphics, realistic physics, mind-bending artificial intelligence, and theater quality audio. Even if the game developer is using COTS components to provide those capabilities, he/she must still acquire a large amount of domain knowledge in order to use the components properly. The simple fact is game developers are forced to become experts in various technical fields when they should be focusing on developing gameplay.

The required level of domain knowledge is only going to increase as game technology advances, and an attempt to resolve this issue must be made soon. This thesis will endeavor to not only identify the commonalities between component interfaces, but also provide a design that minimizes the required component API understanding in order to use a COTS component.

In order to verify the resulting architecture meets this requirement the architecture should demonstrate a reduced API into the component itself. The technology components should also integrate into a game without requiring the game programmer understand the domain in order to use it. This eliminates the possibility of writing technology components a functional libraries.

1.2.3 Architectural Requirement: Flexibility / Modifiability

Flexibility is key to the future of game development. Due to rising production costs, the ability to mix and match re-usable software modules is critical to keeping costs down. The proposed architecture should be game genre and technology independent allowing developers to create a variety of games using various technologies. In order for this architecture to make an impact on the games industry, it must be flexible enough that any game project can use it.

In order to verify the resulting architecture meets this requirement it must be possible to demonstrate that very different games can be written using the final architecture.

1.2.4 Architectural Requirement: Expandability / Maintainability

Another critical architectural requirement, due to rising production costs, is expandability and maintainability. Successful games often have new incarnations with expanded game play and updated technology. For example, Blizzard's™ successful game Warcraft™ is currently on its third iteration with Warcraft™ 3. The new game features added game play elements like powerful heroes and beautiful 3D graphics, but the underlying game is still very similar. A successful architecture should easily allow for this type of game evolution.

In order to verify the resulting architecture meets this requirement it must be possible to demonstrate the architecture can easily support new or updated technology as well as new functionality. For example it should be easy for developers to move a 2D game to 3D graphics without a massive overhaul.

1.2.5 Performance and Other Quality Attributes are NOT Requirements

It may seem odd to not include performance as a key requirement when designing an architecture for a domain that demands such a high degree of performance. The reason for this stems from the belief that performance is far less significant at the inter-component communication level than it is within the subsystem itself. For example, the graphical rendering loop to draw the 10 million triangles of an object is far more significant to performance than the single inter-component communication telling the graphics system to draw the object. Performance will not be ignored in the design process, but the previously stated required quality attributes will carry a higher priority.

Other quality attributes, like reliability or portability, are also not ignored. The scope of this thesis, however, must be limited to qualities that can be verified and validated within the allotted time frame. Follow-up work would be to use the SEI's architectural tradeoff analysis method to determine how these other quality attributes are supported by this architecture. So for the purposes of this thesis, only those qualities deemed most important became a requirement.

1.3 Contributions

The primary contributions of this thesis are the following:

- A better understanding of games as systems. The artifacts created in this thesis will provide insight into what subsystems are involved in electronic games and their boundaries.
- An architecture that supports easy development and integration of COTS components for electronic games.
- An architecture that supports localization of domain knowledge, relieving the requirement for game developers to become experts in everything.
- An architecture that supports flexibility and expandability in game development by allowing developers to easily add/remove/modify game technology components.
- An architecture that support expandability and maintainability allowing developers to more easily expand a game into a future incarnation.

2 LITERATURE REVIEW

2.1 Current State of Game Development in Literature

There are currently dozens of books available on the subject of game development. Most, however, cover in great detail a specific topic in game development rather than an overall architecture. While these books definitely have their purposes, there doesn't exist any literature on how to properly organize these tidbits of knowledge.

Kevin Hawkin's and David Astle's book *OpenGL Game Programming* is a good example of a typical game programming book. The book covers some of the many graphics obstacles present in game development and how to use the OpenGL API. The book discusses 3-D math, lighting, texturing, transformations, and other topics of interest in programming 3-D graphics. After finishing this book the reader will have a solid understanding of graphics and the OpenGL API, but using this knowledge within the context of a complex system such as a game is still a mystery.

While the book is very well written, and covers the technical details involved in pushing pixels with OpenGL, it gives almost no architecture or design information. The book uses examples with a very monolithic design. A single game object will contain everything - graphics code, AI, physics, etc (See Figure 4 below). While this approach is fine for teaching the details of a game feature, it is HUGELY inadequate for a real game. The simple separation of logic at the class level just isn't enough for projects that can reach into the millions of lines of code.

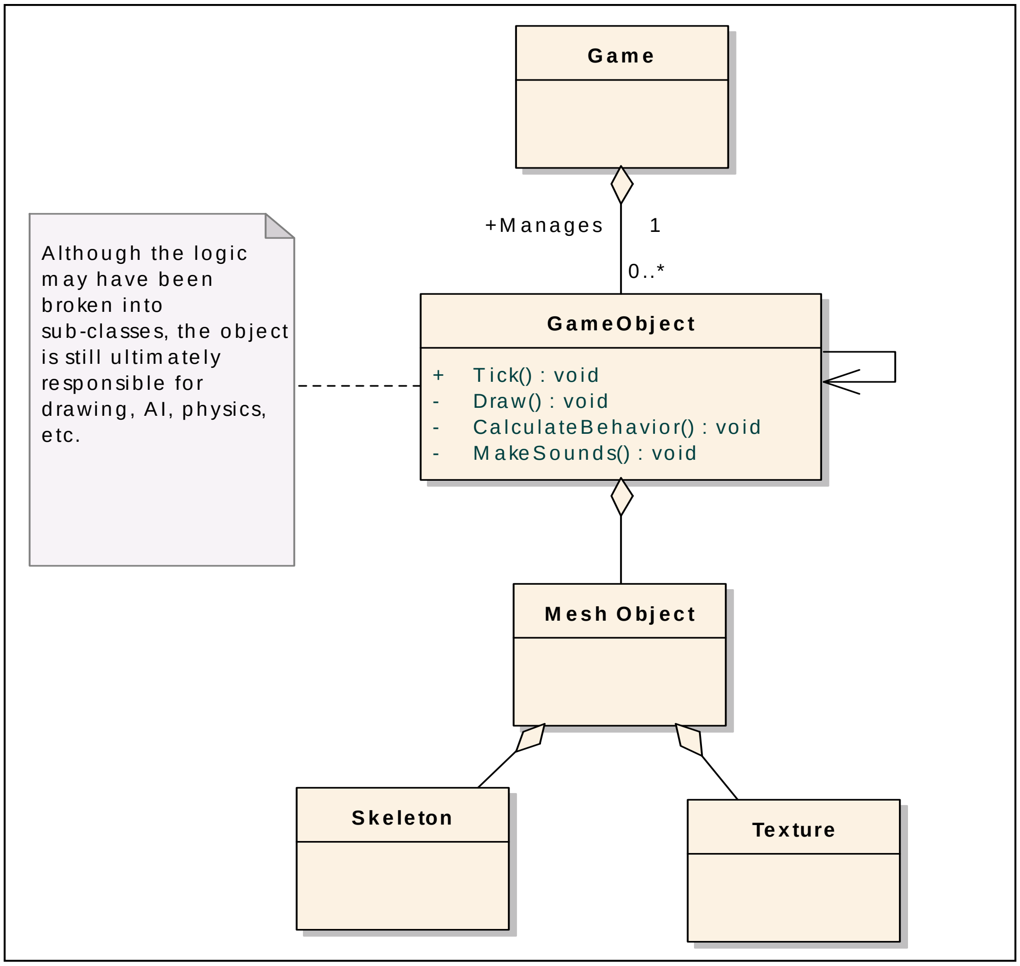


Figure 4 - Object/Class Level Separation of Logic

In order to see the many problems with such a microscopic approach to architecture, consider some of the issues game developers regularly face. First the design gives no insight into issues like portability, a very real concern for businesses interested in the various consoles as well as the PC. Next the code is not re-usable because objects are tightly coupled to their behavior. The design is neither flexible nor maintainable because this design is VERY tightly coupled and changes you make have the potential to affect the entire system.

Rudy Rucker's book *Software Engineering and Computer Games* makes an attempt to teach game development with a reusable architecture. The book creates a "Document/View" game framework. The emphasis is on the framework, as it is possible to create many different games by simply expanding the author's "pop" framework.

The book introduces how design patterns can be used in a game context, and why re-use should be important to a game developer. The author uses the document/view architecture to separate the data from the drawing code, thus allowing changes to the data without touching the visualization code.

While this framework has a great deal of flexibility in terms of game objects, it is still quite limited. AI and physics are still left inside the objects making changes to those areas very difficult. And while the graphics are somewhat separate from the objects, the author still uses direct access between the graphics and the data making the components both very dependant upon each other, and not quite staying true to the architectural model.

2.2 The Latest Book Trends in Game Development

The latest trend in game books is the "gems" like books. Books like *Game Programming Gems* and *AI Game Programming Wisdom* offer developer ready nuggets of wisdom. Snippets of code that offer very good solutions to difficult problems commonly found in game development. These books present low level solutions, usually in the form of a C++ class or two, that solve problems game programmers face everyday.

These books are an incredible resource because almost all their "gems" are architecture independent. They are solutions aimed squarely at helping the programmer, not the system architect. So while the books are an excellent resource to any game developer, the solutions could not be strung together to form a coherent architecture. Developers can use the solution to solve a specific problem, but they may not understand WHERE the solution best fits into the overall system.

2.3 The First and Only Real Attempt at Game Architecture

Andrew Rollings's and Dave Morris's *Game Architecture and Design* is the only book on the market right now that discusses games in terms of their architecture. The book proposes to design around the quote by Dave Roderick, "A game is just a real-time database with a pretty front end."   While that statement might seem correct, this thesis proposes the slightly modified statement – games are a system of systems operating on a database with a pretty front end.

The book gives an excellent introduction into the roots of game development and why architecture and software engineering practices have never really taken hold in this area of software. The authors attribute the lack of engineering practices to the origins and attitudes of game developers. Games originated from solo programmers who hand coded every line, and that solo attitude still prevails in the industry today. Not using third party components is still a point of pride for many developers.

While the authors provide an excellent history of the game development process, the book really doesn't spend much time on architecture (despite the fact that "architecture" is in the book's title).   The book proposes an architecture for a game, but really doesn't provide any insight as to how the components communicate, or even why the proposed architecture is suitable and useful.

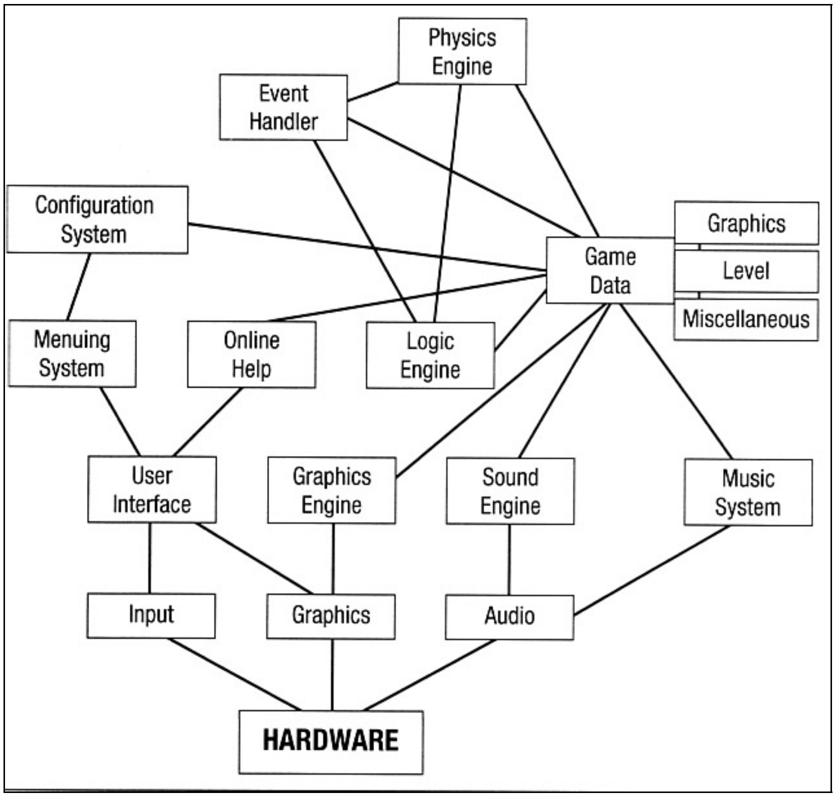


Figure 5 Rollings' and Morris' Game Architecture

2.4 Software Architecture

In order to properly design a flexible and expandable architecture for games, it is not only necessary to understand games, but also software architecture in general. Len Bass, Paul Clements, and Rick Kazman wrote *Software Architecture in Practice* as a very good introduction to software architecture. The book uses clear English to explain what an architecture is, as well as the concepts involved, including architectural style, reference models, and the reference architecture.

Software Architecture in Practice defines many of the quality attributes associated with an architecture, as well as what styles are best suited to each attribute. This book should prove useful when work begins on designing the proposed architecture. The reference offers a great deal of information that should help narrow down the search for architectural candidates.

Another book that provides some very useful insight in designing an architecture is *Designing Flexible Object-Oriented Systems with UML* by Charles Richter. This book provides many simple techniques to identify design flaws that can affect flexibility. The author teaches some guidelines to increase cohesion and decrease coupling, the advantages and disadvantages of class generalization and specialization, and an analysis of specialization versus aggregation. Richter also gives insight in how to analyze dynamic diagrams (e.g. sequence diagrams) for flexibility.

Richter's book should provide the litmus test for the flexibility in my design. He provides an informal, but effective way to quickly assess a design in terms of flexibility. Once the design has passed this informal inspection, a more formal approach can begin and the demo can be built.

3 THESIS METHODOLOGY

This thesis takes a pretty straightforward approach to arrive at the desired architecture. The first step is to analyze and understand games as software systems using standardize software engineering practices. Only looking at a few select games will scale this monumental task down significantly. The analysis will be further limited to identifying the functional modules and their interfaces. This level of analysis should provide enough of an understanding to begin the design work for the architecture.

The next step is to identify candidate architectural styles that have the quality attributes games require as identified in the “High Level Objectives and Goals” section of Chapter One. This step should yield architectural styles that should be considered when constructing potential architectures.

Once the preliminary research has been completed, the architecture design can begin. This involves incorporating various architectural styles into a design until the architecture can support not only the architectural requirements from Chapter One, but also the functionality Identified during the analysis phase. Through design trial and error, and architectural analysis techniques a proposed architecture should emerge.

After the proposed architecture has been designed, it is time to prove that it can work. The first step to proving the architecture will be to apply the architecture in the form of a simple design to the analyzed games. This will help to validate that the architecture can support the types of systems it was intended for. The next step is to actually build a game-like system to demonstrate the quality attributes. Unfortunately designing a commercial quality game to fully demonstrate the capabilities of the architecture are beyond the scope of this thesis. A demonstrative subset of game functionality, however, will be put together into a prototype to show some of the more important features. A prototype will also have the added benefit of helping to refine the architecture into a more correct state, as well as identify some of it's limitations.

3.1 Analysis of Games as Software Systems

In order to design an architecture for the games of tomorrow, we must first understand the problems faced today. As noted in the literature review section, there exists very little documentation on the subject of architecture in games. Since more information is required, more creative approaches to analysis will be taken.

Since actual documentation on architecture in existing games is virtually non-existent, we will do the next best thing – understand the design of a game similar to existing games. The approach is simple. Treat an existing game as the customer requirements, and attempt to design a game that meets those requirements following standard software engineering practices. Performing this process for several games should provide a satisfactory understanding of what is required in an architecture to meet the needs of those existing games.

3.1.1 Selecting Games to Analyze

Since the goal of this thesis is to construct an architecture that will meet the needs of most electronic games, more than one game must be analyzed. In truth, such an architecture would require a thorough understanding of every possible game created. Due to the constraints of a temporal existence, this thesis will attempt to refine the search space into something more manageable.

Rather than analyze every existing game, existing games will be divided into categories where a single title could be selected to represent all games in that category. Fortunately the electronic games industry has already categorized titles into genres and we merely have to locate games representative of their genres. This approach should provide the best possible results given the time restrictions.

Game genres can be further divided into sub-categories like single-player vs. networked, 2-D vs. 3-D, etc. but I propose to show that these subdivisions are expansions of the same architecture. For example the differences between a 2-D game, and a 3-D game of the same genre should be localized in the components. However, the types of components and their interactions should remain the same. In the end I hope to show that a single architecture is capable of supporting all these genres.

3.1.1.1 Existing Game Genres

- **Fighting**

  The market was successfully introduced to fighting games in 1991 by Capcom and Street Fighter II. The opportunity to have fantastic heroes battle in hand to hand combat gave adolescent gamers the opportunity to connect to unique alter egos, and began the "golden age" of the arcade ("History of Arcade Games"). Fighting games are among the most simplistic in nature. They are meant to be simple, fast, and fun.

- **First Person Shooters (FPS)**

  First person shooters were invented in the 1992 by John Carmack and ID software with Wolfenstein 3D™ ("A Brief History"). The game ushered in a new era of 3-D immersive worlds where players could explore, and experience the electronic universe in the first person.

This genre is probably the most diverse with games ranging from single player only, shoot to kill everything games like Doom™ and Quake™, to massively multiplayer universes like Halo™. First person shooters are almost always state of the art in terms of technology, and best noted for their outstanding graphics. Releasing a FPS using last years technology is a recipe for disaster in the retail market.

- **Platform**

Platform games are the definitive arcade games. Icons like Super Mario Bros. ™ and Donkey Kong ™ were among the first to dominate the scene. Platform games require the player to navigate a character through various puzzles using a player's wit and skill with the joystick. Platform games are a relatively small market on the computer, but they still dominate the consoles with memorable characters like Lara Croft ™.

- **Strategy**

The electronic strategy games of today are simply extensions of their board game ancestors. Strategy games typically involve intricate rule systems where player must master tactics and strategies rather than fast reflexes. Games range from the 2-D turn based classic Civilization™ to the 3-D real time masterpiece Warcraft™ III.

- **Role Playing**

Role playing is another genre that has its roots outside the electronic forum. Role playing games are a form of interactive fiction, where the player gets to play the role of one or more characters in the story. One of the staples of role playing games is character advancement. The character(s) the player controls will continue to grow in skills and abilities allowing the player to evolve a truly unique alter ego.

- **Sports**

Simply put, sports games are just the electronic versions of the real thing. Electronic sports games allow gamers to play the game without actually having train there whole lives to become professional athletes. Unhappy with the outcome of the super bowl, challenge your neighbor to a rematch in Madden 2002™.

Obviously there are games that do not fit into any of these genres or would be better described as a combination of genres, but these six categories arguably represent the bulk of electronic games available today.

3.1.1.2 Further Refinement – Isolate Important Properties

Unfortunately, properly analyzing even 6 games is too large of a task for the scope of this thesis. To ensure this further scaling has a minimal impact on the quality of the resulting architecture, I've decided to isolate the most important features. A minimum selection of games that covers those features will be chosen.

- **2D vs. 3D**

2D games are two dimensional games where the character exists in a two dimensional world. Platform games like Super Mario Brothers™ and strategy games like Starcraft™ are examples of 2D games.

3D games are games that take place in the third dimension. Here the distinction must be made between two dimensional games using 3D graphics, like Warcraft ™ 3 and games with fully three dimensional worlds like Quake™. For this thesis it is important to select a game in the later category, because it is important to maximize the differences in the game components. Fully three dimensional worlds require different physics, AI, as well as 3D graphics.

All games fall under one of these two categories, so the final selection must include one game from each category.

- **Non-Networked vs. Networked vs. Massively Multiplayer**

Non-networked games are games that exist on only one machine. Code and data does not need to be distributed across a network while the game is playing. Almost all games offer this style of play, allowing the human player to compete against computer opponents on a single machine.

Networked games are games where human players can compete against other human players over a network. Most games of this sort use the simple client/server model and usually have a set maximum number of players (clients) per game.

Massively Multiplayer Online Games (MMOG) have been around for a while in many text based multi-user dungeons or MUDs, but have become very popular in the mainstream with the 3D dungeon romp - Everquest™. MMOGs allow thousands of players to exist persistently in a virtual world. Unfortunately due to the scope of this thesis, MMOGs will not be covered, but definitely represent an area that should be covered in future research.

- **AI – Single Entity vs. Managed or Team**

  Artificial intelligence in games can be broken into two very simple categories. Games with single entity intelligence are games where each game object has its own AI and behaves relative to its own situation. There is no mastermind or general coordinating the actions of the objects to form an overall strategy.

  Managed or Team AI games expand on the single entity AI model and add the concept of collaboration between objects. Objects still have their own intelligence, but a new layer has been added that can view the game in terms of tactics and strategy.

3.1.2 The Selected Games for Analysis

After a great deal of review, the search has been narrowed down to two games that exist in to different genres and cover all the important properties. While these two games cannot fully represent all possible electronic games, these two games should provide a solid foundation given the time constraints and scope of this thesis. This foundation should be adequate to isolate many of the component interactions and support the design of an architecture that could support the needs of most games.

<u>Starcraft™ by Blizzard Entertainment</u>

The first game chosen for analysis is the award winning Starcraft™ by Blizzard Entertainment™. The game features a 2D isometric view and some of the best game play ever. The game was released in 1998, and has become the yard stick all other real-time strategy (RTS) games are measured by. For analysis purposes the game was chosen because it is two dimensional, offers solid non-networked or single player game play, and a managed AI system.

Figure 6 - Screenshot from the Game Starcraft

Unreal Tournament

Unreal Tournament(tm) by Epic Games Inc. is easily one of the best networked first person shooters ever. The game offers up to 16 players a chance decimate each other in a futuristic combat arena. Players enter UT's 3D proving grounds and become the combatant, taking control of a single character. While newer iterations of UT have been developed since UT was released in 1999 (“Unreal” 1), see screenshots below, they are primarily upgrades in technology. Unreal Tournament(tm) was chosen because it features 3D graphics, networked play, and any AI is primarily centered around a single entity.

Figure 7 - Screenshot from Unreal Tournament

Figure 8 - Screenshot Unreal Tournament 2004

3.1.3 Analyzing the Games

Having selected a seemingly diverse pair of representative games, we can begin the analysis process. By designing an architecture capable of supporting these two dissimilar games, it is the hope of this thesis that the architecture can support the development of just about any type of game. The analysis will pretty much follow the standard software engineering practices for system development.

The process begins with understanding the systems requirements, which can be done by treating the final game as the customer requirements. From the final game, use cases can be derived and reviewed for further analysis. From that point, we can begin to find the subsystem interactions that need to exist in the proposed design.

3.1.3.1 Analyzing Starcraft™ Requirements with Use-Cases

The first part of analysis is to understand the requirements of the system we are trying to build, or in our case merely understand. Since our requirements are based on a finished piece of software, requirements and use-cases can be harvested from the game's manual and from playing the game itself. After a first pass of studying the manual and actually playing the game, I came up with the following use case diagram:

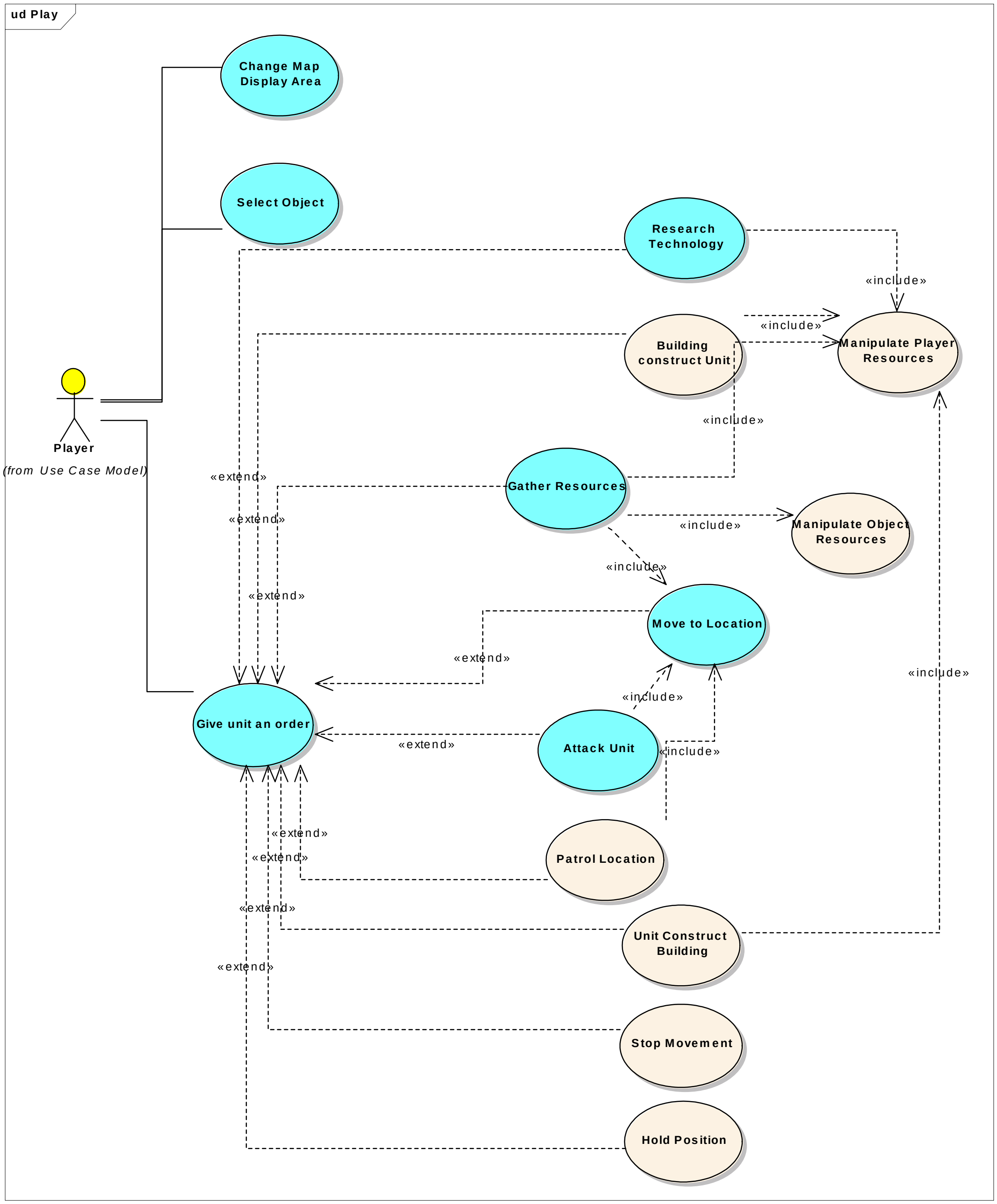


Figure 9 - Playing Starcraft Use Case Diagram

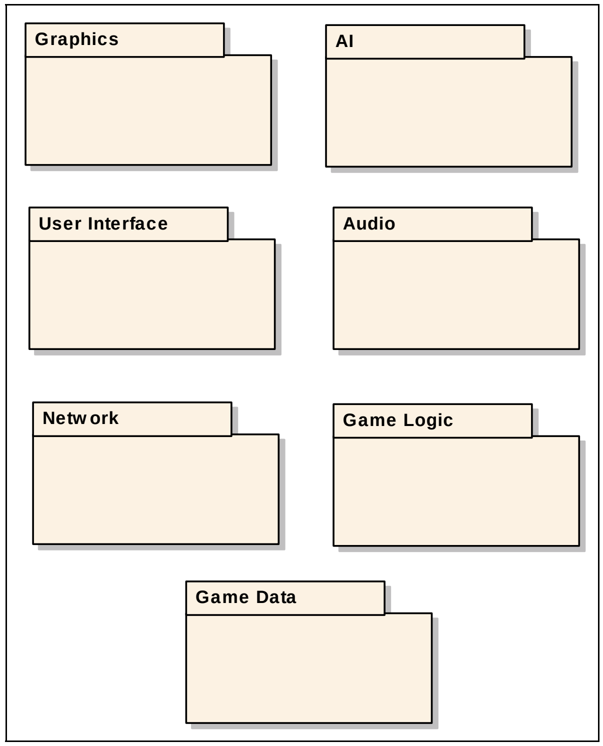


Figure 10 - Logical Modules

Based on the details of the use cases in the diagram in Figure 9 above, we can begin to identify the functional modules involved in the game of Starcraft™. 2D graphics functionality is needed to render the game objects, the user interface functional module will capture the players input, and so on. While Figure 10 above is not meant to show the physical separation of subsystems as a component diagram would, it does show at a high level what kinds of functionality are needed within the game Starcraft™.

Before we begin analyzing the types of sub-system interactions that need to exist in the system, however, we must first isolate which use-cases will be used to guide the analysis. The final analysis of this game, located in appendix A, has very many use-cases. Due to the time and scope constraints on this thesis, it would be impossible to fully explore them all. To ensure the research is still adequate I based the selections on some very simple criteria.

First the use-case must be fundamentally important to the game. Since our original game title selection was based on each game being representative of many other games, there is no point wasting time analyzing actions that don't represent those other games. Second, the use-case must require multiple sub-systems to collaborate. Since the goal of this next phase is to understand logical module interactions, we can eliminate the trivial use-cases. In all diagrams, use-cases selected for further elaboration have been colored a light blue.

3.1.3.2 Understanding the Sub-System Interaction

Having selected multiple use-cases for further analysis we can begin trying to understand the communication between logical modules required to realize those use cases. Consider the *Select Object* use-case from Figure 12. "User left-clicks the mouse button while the cursor is placed directly over a selectable object in the main view. The selected object is marked with a green circle and is ready to receive orders." In applications using the "document/view" architecture, this use-case is almost trivial to implement. The view receives the mouse click, determines which object was clicked based on its screen coordinates, and sends the click event to the object for processing. The object can then decide to draw a circle around itself or whatever.

At this stage we have not yet decided anything further about the architecture, so the focus is not to design the interactions as in the document/view example. Instead the goal at the analysis phase is to understand the "kinds" of interactions, not how those interactions will actually be designed. Consider that same *Select Object* use-case following a model-view-controller approach. The "kinds" of component interactions that need to take place are still the same.

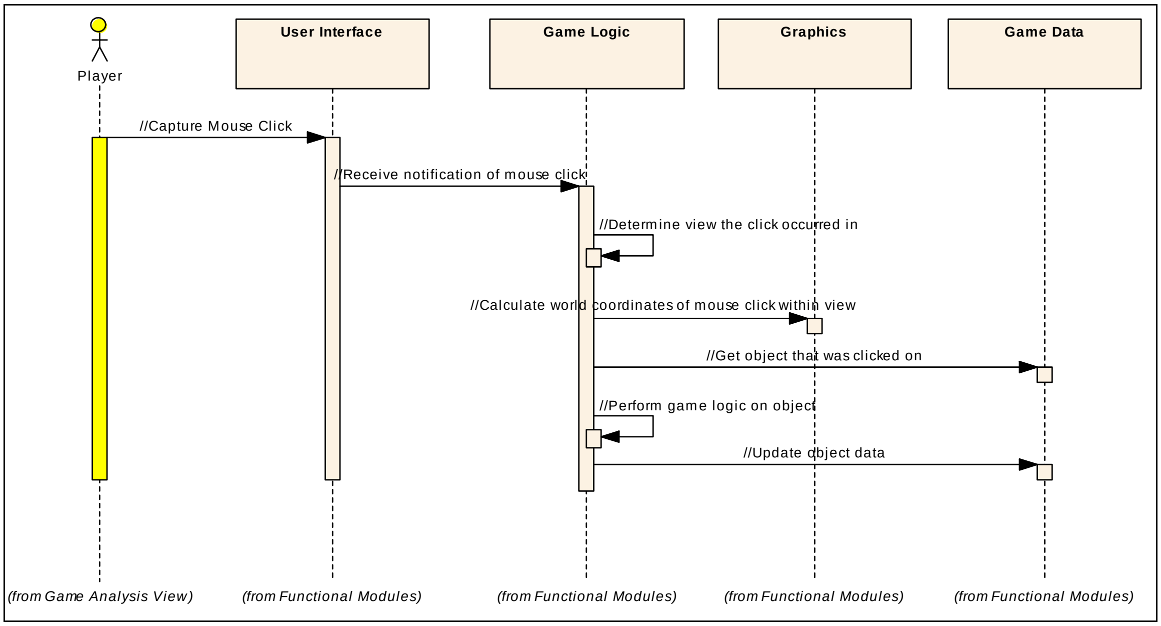


Figure 11 - Select Object (Subsystem interactions)

Figure 11 above shows an example sub-system interaction. The trick is to understand that the diagram above is NOT the design. The "Select Object" sequence diagram above shows that the graphics system is involved in routing mouse clicks to the proper object. It does not necessarily mean the Game Logic system calls the graphics system. Perhaps current screen position is part of the object data set by the graphics system at a different time. The important thing to note is that in order to determine which object was clicked, the UI and graphics systems are involved. Once all the important use cases have been further analyzed to isolate the kinds of interactions we can begin creating a potential design. To see other component sequences refer to appendix A.

3.2 *Identify Candidate Architectural Styles*

The next step before we can design an architecture is to consider the architectural styles that have already been shown to exhibit the quality attributes games require. In its simplest form an architectural style is a set of components, their constraints, and the constraints on their communication (Bass et al. 25). By incorporating well-understood architectural patterns, we are more likely to achieve a hybrid design that will achieve our goals.

Several architectural styles appear to have some of the desired quality attributes, and will be reviewed.

3.2.1 Layered

The layered architectural style divides system functionality hierarchical layers where each layer provides services to the layers above and below it. The layered approach tends to promote re-use by keeping the application specific code at the top most levels, allowing developers to re-use the framework below (Duffy). Re-use is directly tied to the flexibility and modifiability requirements of this thesis.

3.2.2 Data-Centered

The data-centered style is essentially a centralized data store with independent clients connecting to operate on the data. Data-centered styles offer an easy way integrate different systems because the clients are independent of each other, and the data store is independent of the clients (Bass et al. 95-96)

3.2.3 Independent Components

Independent processes communicating via messages define the independent component architecture. Components register the kinds of information they can process, and communicate through messages (Bass et al. 101-102). One interesting advantage of the independent component styles is that all components need not exist. The decoupled communication system is such that published messages may not have any subscribers. A well-designed system could add and remove functionality at will.

3.2.4 Data Flow

Data flow architectural styles like pipe and filter tend to offer a great deal of re-use, and are generally easy to maintain and expand (Calvert). By focusing on incremental transformations of data, systems are very simple to understand and change. Systems can be easily expanded or modified simply by plugging in new or different data processing components (Bass et al. 96-97). The notion of effortlessly expanding games by extending the chain of data processors is very appealing.

3.2.5 System of Systems

The system of systems architecture is the part of engineering work being done to integrate multiple complex systems. The SoS approach is interesting because both games and enterprise applications must integrate systems of entirely different domains. Graphics, physics, AI, etc. are entirely unique domains being used together in a single application. Another interesting aspect of SoS is the point of view that a system is emergent from the integration of the individual subsystems ("Definitions"). In other words a game is the result of integrating an AI system, a physics system, etc.

Such a unique view matches one of our initial requirements of domain knowledge localization. So if a graphics engine is a complete system, and the game is actually the result of the graphics engine working with other systems, it may be possible to keep the graphics details hidden from the game itself.

3.3 Architecture Design

At this point both games selected for study have been analyzed such that we have descent understanding of what logical modules exist, and the kinds of interactions that must occur to perform the game functionality. The next step is to actually determine the overall system layout, and how the different subsystem interaction will occur. By incorporating the various architectural styles noted in Section 3.2 of this thesis, we will design an architecture that should meet the requirements we have laid out, as well as support the functional needs common to games.

3.3.1 Choosing a Topology

The first step to developing an architecture is deciding upon a topology. The topology is the over-all layout of the system, and has significant impact in terms of modifiability and reusability. The topology determines what logical systems are connected; thereby setting what coupling may exist. The plan is to see how different architectural styles can be applied to the logical modules in games, and determine the affects it might have on the quality attributes the desired architecture requires. While only one topology will be selected (or perhaps a hybrid of a select few) for further study, those that weren't selected may provide some ideas that can still be incorporated into the final architecture.

3.3.1.1 Layered Architectural Style

The first major topology considered was the layered architectural style. The layered architectural style tends to offer many quality attributes, of which re-usability and modifiability are most import to our goals. Looking at the simple diagram in Figure 12 below, the game specific code is localized in the top-most layer. By localizing the game specific logic to a single layer, new games can be created re-using the layers below.

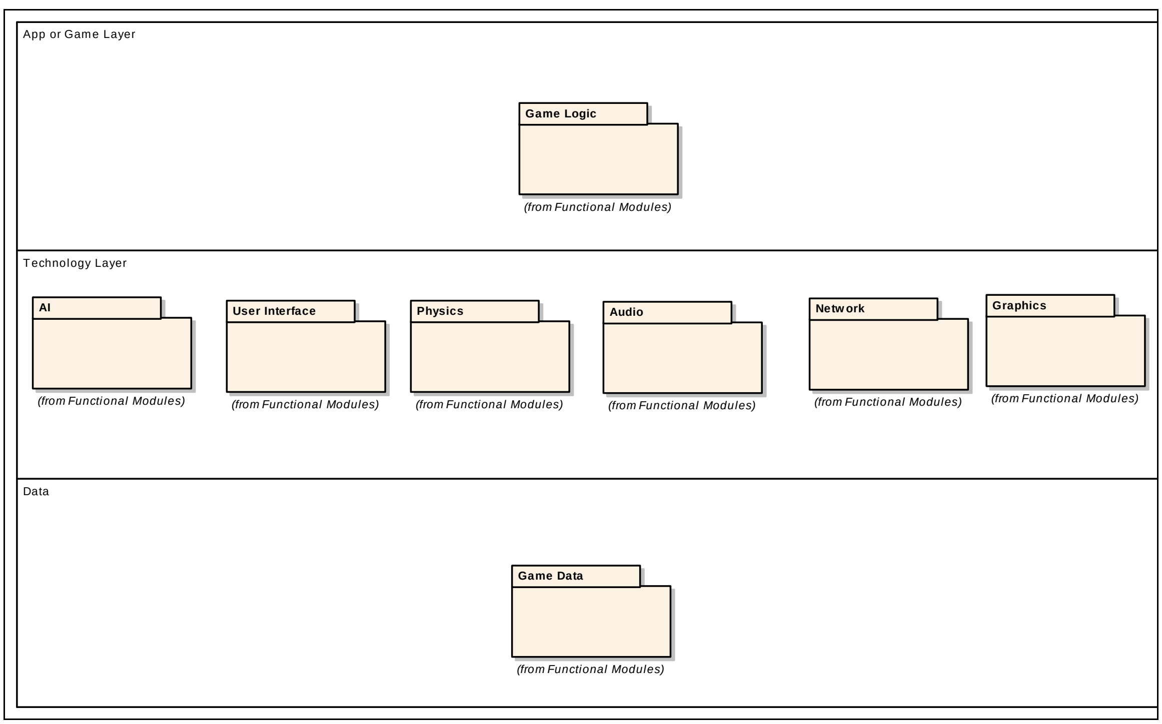


Figure 12- A Simple Layered Architecture

The above "start" of an architecture has many problems that ultimately led to the dismissal of this topology. First, while the architectural approach offers some re-usability between games, it does not appear to offer much re-usability between different types of games (i.e. different technologies). This view of a layered approach doesn't offer much insight in how a developer might move from a 2D platform game, to a 3D first person shooter. Such a change would require a very different graphics module, a very different AI module, as well as requiring additional modules that probably wouldn't exist in a 2D platform game (like physics).

Obviously this approach could be refined with layers further divided, but the underlying problem still exists. It isn't just the game code that is likely to change, but the technology modules as well. Also due to the fact that different sets of logical modules may be needed for different games, with potentially different module interactions, perhaps layering does not isolate likely changes in the best possible way.

3.3.1.2 Data Flow Architectural Style

Data flow architectures offer a great deal of flexibility in that data processors can be added at will. The problem becomes very apparent, however, when you look at the game modules in this layout (see Figure 13 below). Game components operate on very different data. The data pipe connecting these logical modules would have to be so broad that each module might spend significant overhead parsing and filtering out the large amount of data that isn't used (Calvert).

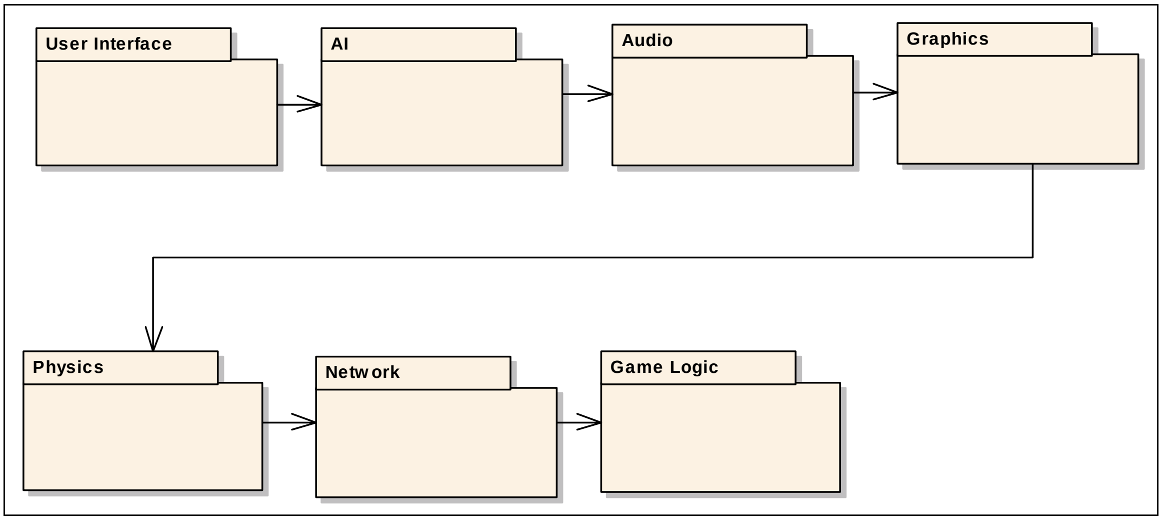


Figure 13- Data Flow

The data flow architecture does, however, present some interesting options for the architecture at the component level. Figure 14 below shows a simple example of how an AI component could be implemented using the data flow architectural style. Unfortunately this thesis is focusing on the architecture at the game system level, so this concept will be left for future research.

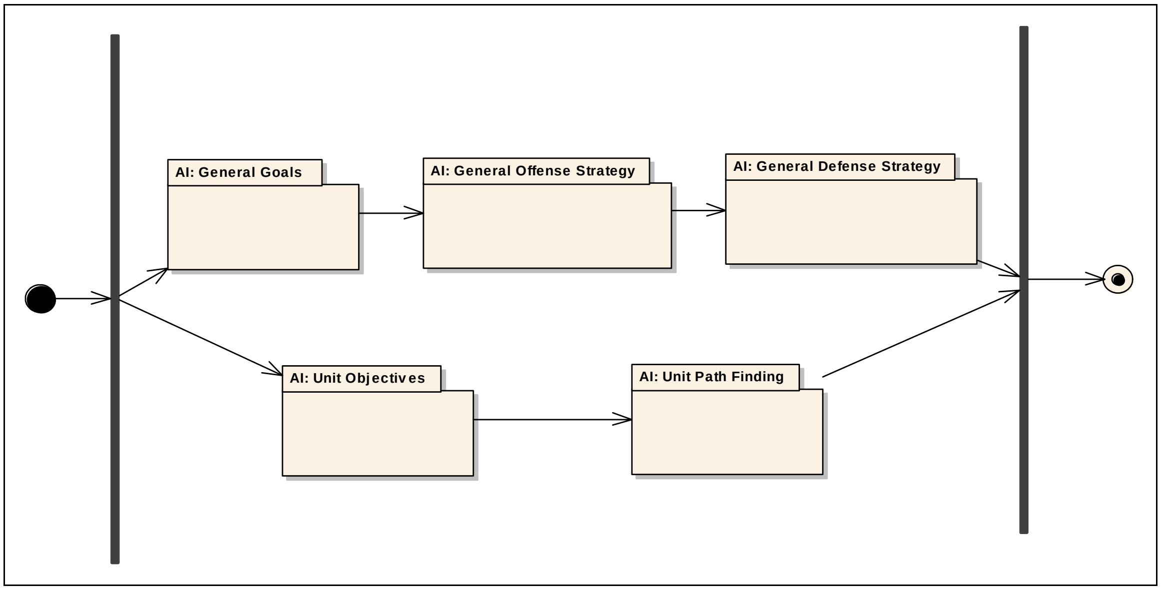


Figure 14- Data Flow at the Component Level (AI)

3.3.1.3 Data Centered Architectural Style

Another major topology of interest is the data centered architectural style. A data centered approach is typically used to create data integrability. Functional modules are less strongly coupled, but often at a cost in performance (Bass et al. 96). The data centered approach minimizes many of the risks identified in the layered approach. First, the logical modules do not have any direct interaction with each other mitigating the issue of changing technology. Changing from a 2D graphics logical module to a 3D graphics logical module should not break the workings of the other sub-systems.

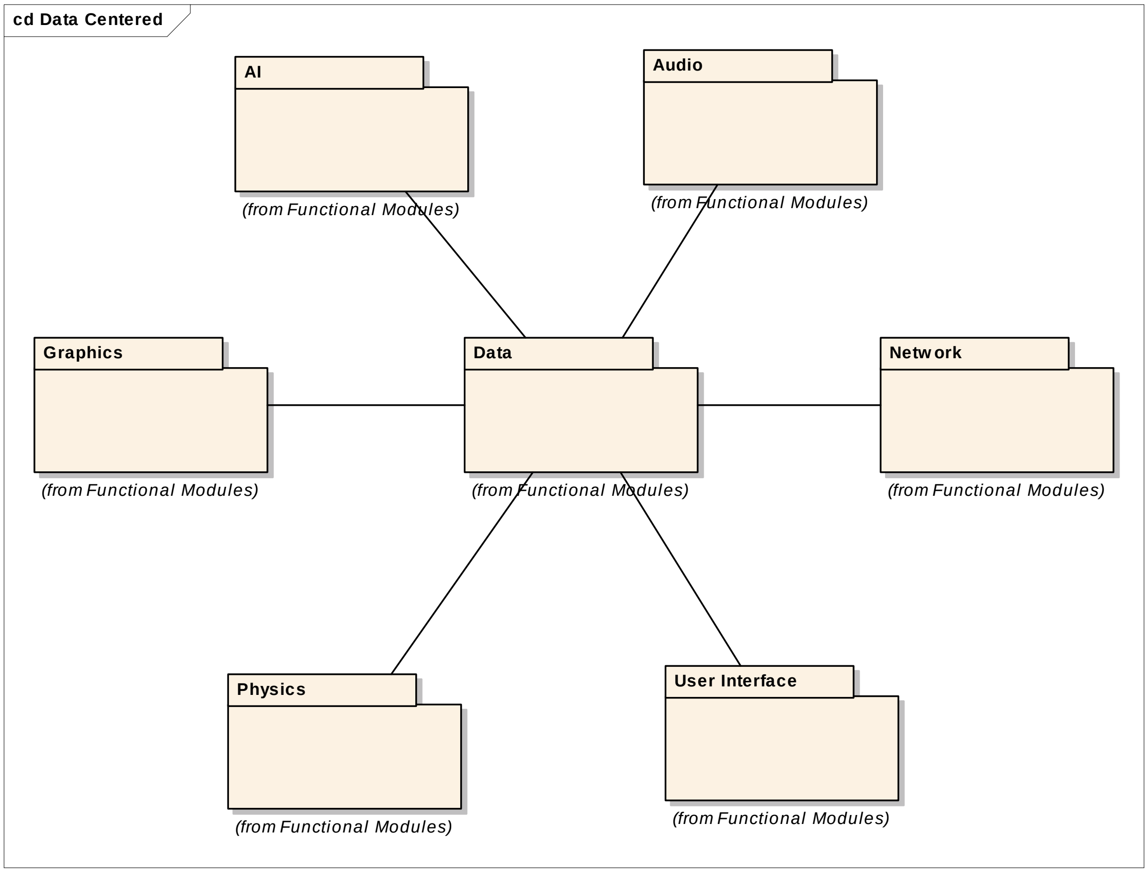


Figure 15 – Data Centered

There is still the issue of modifiability at the game level. Figure 16 above does not give any indication of how the developer can minimize the amount of change when moving from one game to another. In the layered approach, game specific code was localized to a single layer, making it easy for developers to move between similar game projects, while the data centered topology doesn't provide much insight as to how game specific code could be localized. This issue will continue to be worked as the architecture is further fleshed out.

While using a data centered approach does offer many architectural benefits, it drastically impacts how game functionality can be achieved. Consider the use case diagram shown earlier in Figure 11 - Select Object (Subsystem interactions). This simple act of clicking a button changes dramatically because there is no direct association between the User Interface and Graphics logical modules. Both figures demonstrate the same functionality, but the data centered topology has placed some constraints on the way it can be realized.

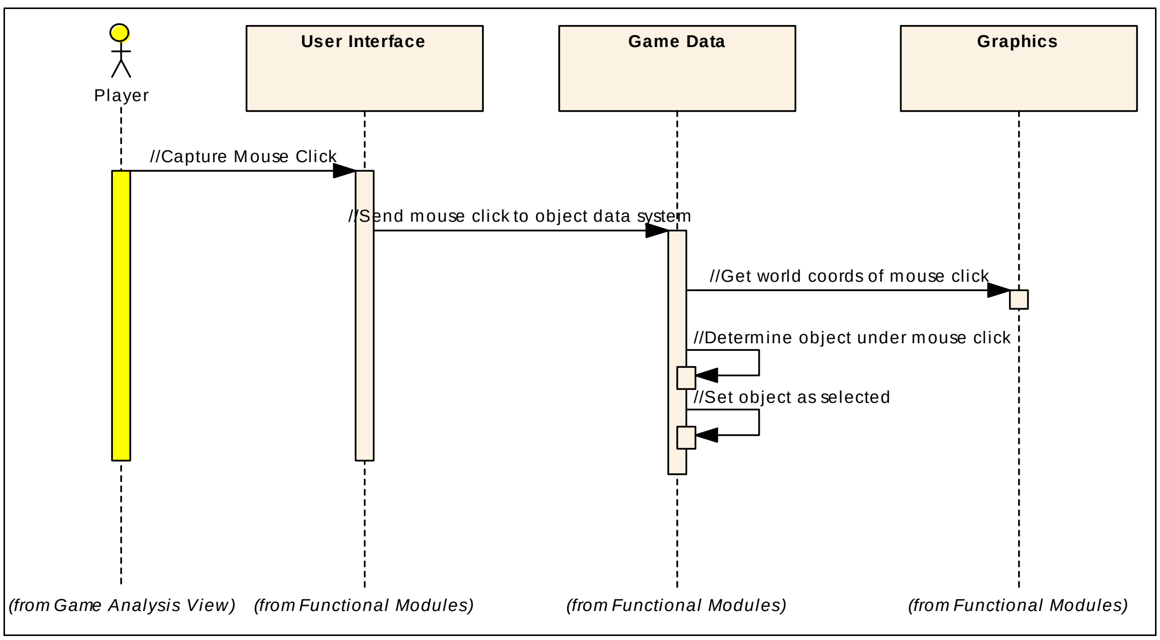


Figure 16 – Select Object (Logical Module Interactions – Data Centered)

3.3.1.4 Independent Components Architectural Style

A game using the independent component architectural style can have any arbitrary topology because of the style's restrictions on communication. Regardless of the layout, independent components remain decoupled because they communicate via messages rather than making function calls. The interesting aspect of this approach is components don't know who they are sending to, and don't necessarily need to wait for a response. This not only means new components can be added/replaced, but partial systems can be built with components missing. This means systems can be put together even before all the components are built, greatly increasing the ability for individual components to be independently developed.

Figure 17 below shows an example of how a game could potentially be put together using the independent component architectural style. This rough sketch highlights some of the strengths and weaknesses of this approach for a game system. The user interface component as an independent component communicating via messages makes perfect sense because user interactions really are asynchronous events. When you start to move to how other components like graphics and AI interact with data, event based communication makes less sense. Every cycle some game data must be drawn, must perform AI, and must have some form of game logic applied to it. The overhead of routing and translating messages becomes significant when the number of messages approaches some threshold. Due to the sheer volume of data involved in games, and the synchronous nature between some of the subsystems and the data, perhaps independent components is not the best architectural style for this domain. It would, however, be an interesting research project to see just how much the messaging overhead would affect systems with synchronous interactions like games.

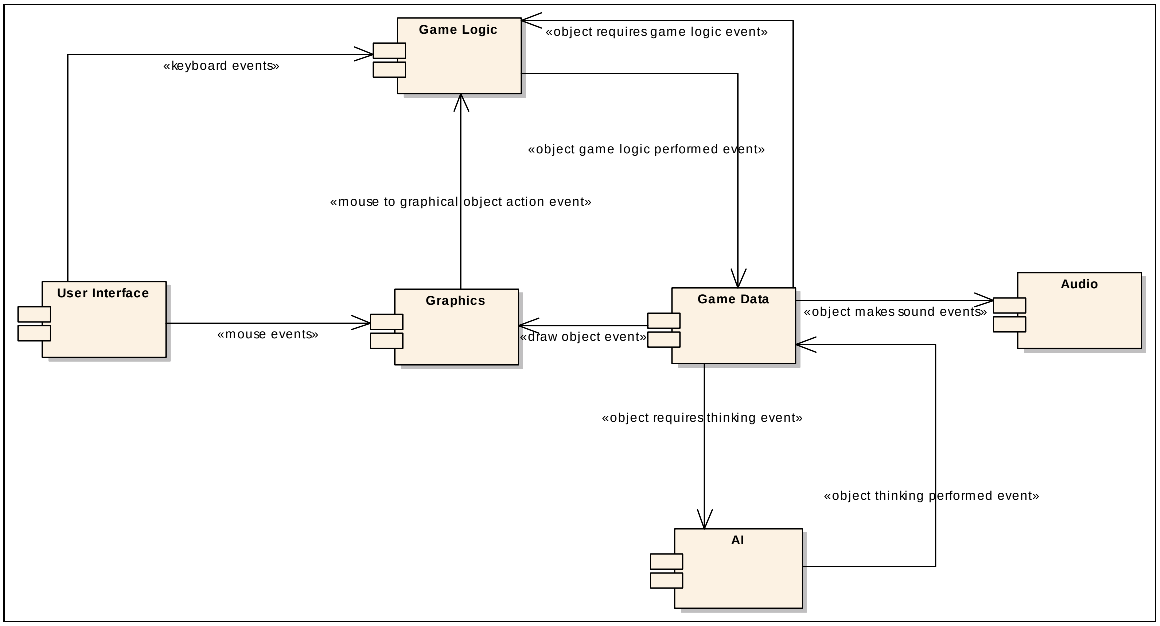


Figure 17- Independent Components

The qualities achieved by independent components should not be completely discarded simply because this particular style may not be the best choice. The ability to put together an incomplete system with components missing is a very useful idea. Consider a development scenario where the graphics system has not been selected, or is behind schedule. An incomplete game system consisting of the game logic, data, and AI could continue to be worked. So even though one of the subsystems cannot be used, the game as a whole can continue integration work.

3.3.1.5 System of Systems

The system of systems was rejected for the same reason the independent components architectural style was rejected. Event based communication is just too inefficient for some of the interactions. The time researching the system of systems perspective, however, was definitely not wasted. The notion that the desired system, a game in our case, can be emergent as a result of the collaboration of other systems is a very interesting idea (“Definition”). Just because the arbitrary topology and method of communication are ineffective for this thesis, doesn’t mean the underlying idea can’t be used.

3.3.2 Making the Topology Choice

Choosing a topology forms the structure from which the architecture will evolve. It determines how systems can grow and change, and has significant impact on the qualities the final architecture will exhibit (Bass et al. 105-107). The research has shown that arbitrary topologies appear to place too much overhead on communication in order to keep the subsystems truly independent, a key requirement for this thesis. Even though performance was not one of the key requirements for this architecture, other approaches are still able to meet the requirements without imposing such a high performance cost.

The data flow architecture appears to not be the best choice because the logical domains are just too different. Trying to design a universal data pipe for all the data involved in games doesn’t seem like the correct approach. The analysis performed seems to suggest that the data flow architectural style is just the not the best starting point for the system level of abstraction.

The layered approach is more structured and could possibly provide better performance than the other less structured topologies. The layered topology, however, cannot easily abstract functionality in a way that minimizes the effects of changes in technology. Part of reasoning behind this thesis is the belief that changing technology has a greater impact on work required and the level of modifiability, than the ability to swap out the coded game logic. Game technology is moving at an astounding rate, and gamers often only buy games with the latest technology. This means developers must constantly upgrade the technology modules or suffer in sales. It would be possible to place each logical system in its own layer, but doing so essentially emulates the data flow architecture and all its problems.

Ultimately, the data centered topology was chosen for further analysis because it showed the greatest mix of flexibility and performance. The other approaches may have offered some truly desirable characteristics, but had significant inherent disadvantages that would be difficult to overcome. The data centered approach still allows for sub-system independence, but allows a direct communication to the game data. At this point it seems the data-centered layout offers the best chance at designing an architecture that meets all of the proposed requirements.

By moving forward with the data-centered topology this thesis is placing a higher priority on providing flexibility in technology usage than on re-using an existing framework. This prioritization is also matches one of the original goals for this thesis – supporting COTS-based development. Modern game complexity is just too large for single development house to create it all. An architecture design that supports easier integration of COTS technology will likely better serve the industry. It should also be noted that choosing to start from a data-centered topology does not necessarily restrict the use of a different topology at a different level of abstraction. For example, it may be possible to use both a layered and data centered topologies as in Figure 18 below.

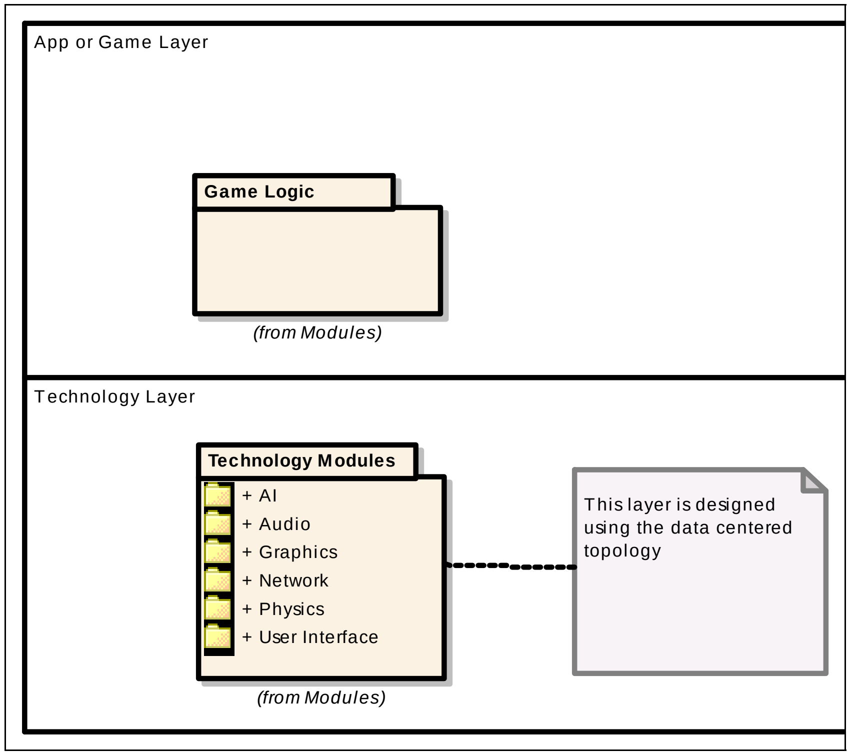


Figure 18 - Layered and Data-Centered

3.3.3 Choosing a Style of Communication

Once the overall topology of the logical modules has been created, the method of communication must be designed. We have decided what modules can communicate, but it has not been decided how that communication will work. Following with the data centered topology there are two common models available for the communication between clients and the data.

3.3.3.1 Repository

The first is the repository model where data resides in a passive repository. Because the data repository is passive, clients are responsible responsible for pulling the data and determining if it has changed. The repository is probably the simplest to understand because the data store is essential a database answering queries.

The only real downside to this methodology is the increased traffic between a client and the data store. The client is requesting data for processing even if the data has not changed.

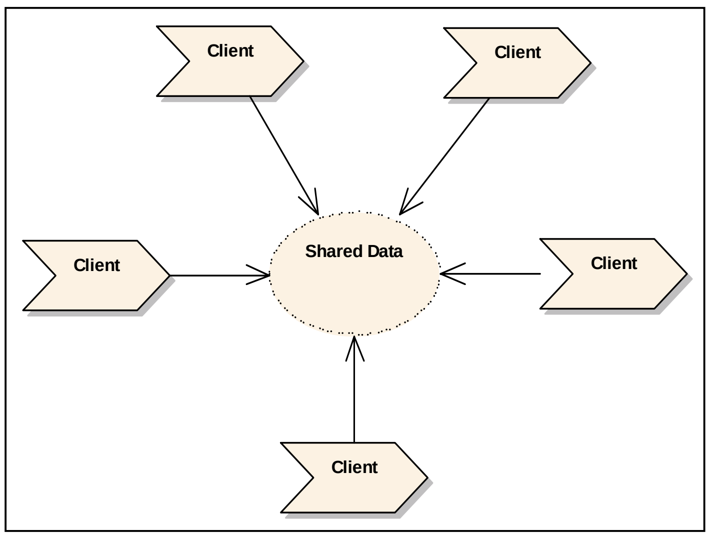


Figure 19 - Repository

3.3.3.2 Blackboard

The second common model for communication is the blackboard method. In this model the data repository is active and sends update messages to clients informing them of the updates to the data (Bass et al. 95). The blackboard methodology is an attempt at reducing the amount of communication and as a form to keep the clients synchronized.

3.3.3.3 Making the Communications Choice

For the games domain the repository model was chosen because it makes the most sense logically. First, the increased communication between the client and the data store is less of a concern because both will likely reside on the same computer. Second, a domain-specific module will need to operate on an object whether it's data has changed or not. For example, a graphics engine will need to draw a visible object even if it's position hasn't changed since the last time it was drawn. Lastly, the repository model also has the benefit of keeping all the data localized in the one area meaning domain-specific modules don't need to maintain local copies of the data.

3.3.4 Synchronicity

Synchronicity is how the data and control flow through the functional modules. Because synchronicity is tightly tied to the topology and method of communication we have already eliminated some possibilities. For example, since we have chosen not to use the blackboard method of communication asynchronous methods of synchronization may not be the best choice. Fortunately traditional approaches to game development already use a method of “ticking” game objects, proving that games can be built using a synchronous approach.

3.3.4.1 Synchrous at the Object Level

Synchronization at the object level is where all of the object's functionality is completed before moving on to the next object. In other words an object performs AI, draws itself, etc. then moves on to the next object. If you stick with the paradigm that a game is just a bunch of game objects then this method makes sense.

3.3.4.2 Batch Synchronization

Batch synchronization is the case where a large group of objects are processesed completely before moving on to the next group. A game example might be that all objects perform their AI calculations before they are drawn. This approach starts to make sense the more complex the specific functionality becomes.

3.3.4.3 Hybrid Synchronization

Current approaches to game development today often use a mix of synchronous approaches at the object level and component level. “Ticking” an object may result in the object performing AI, and making sound, while drawing the objects may be done as an entire batch. This probably a result as games evolved. In early days, games were simple enough that synchronization at the object level. As technology has grown more complex, it's often easier to write an entire “engine” to perform things like graphical rendering as a batch operation (Rollings 453-454).

3.3.4.4 Making the Synchronicity Choice

The choice to move ahead with batch synchronization was made for several important reasons. First, synchronization at the object level using the data centered topology with a passive repository does not make a lot of sense. Synchronizing at the object level defeats the whole purpose of having functional modules operating independently around a common data store. Having each functional module operate on the relevant objects and then moving to the next functional module does. Second, one of the main reasons for this research is to deal with the fact the domain-specific processing is becoming more and more complex. And as games are already beginning to see, it is easier to handle complex calculations when operating as an “engine” performing a specific type of functionality all at once.

3.4 The Idea – System of Systems Philosophy

Having performed a great deal of research in both games and software architecture I have come to really like the system of systems philosophy. While the common concept of SoS appears to have too many performance issues to make it viable for games, the underlying idea is sound. The notion that that independent and complete systems are collaborating and result in an emergent system is very powerful.

Designing a game as a collaboration of independent game subsystems has a great deal of potential. First, development and test are simplified because dependencies between sub-systems are eliminated. Second, incorporating the subsystems into a game can potentially become much, much simpler. Because a logical module is a complete system, the game is not using the module as a programming library with game specific function calls. Instead the logical module is configured to behave as a system that will result in what the desired game needs. So using a game subsystem becomes a matter of configuring a system, rather than learning and using a domain-specific programming API. The proposed architecture will attempt to incorporate this simple idea, and possibly create a new approach to developing games.

4 THE PROPOSED ARCHITECTURE (and a Simple Design)

The proposed architecture takes a step back from looking at games as a system of game objects, and looks at them more as a data centered System of Systems (SoS). An architecture where external systems (graphics, AI, etc.) work together toward a common goal, and the game is formed as the collaboration between those systems working on the same data set. This chapter will present the architecture and a simple design using the proposed architecture.

4.1 The Data-Centered System of Systems Topology

The architectural structure is represent below in Figure 19. Domain-specific systems operate independently on a shared collection of data. The domain-specific systems are responsible for requesting data to operate on, and update. Another issue to note, but will be further explained, is the domain-specific systems can store domain-specific data related to game objects within the common data store.

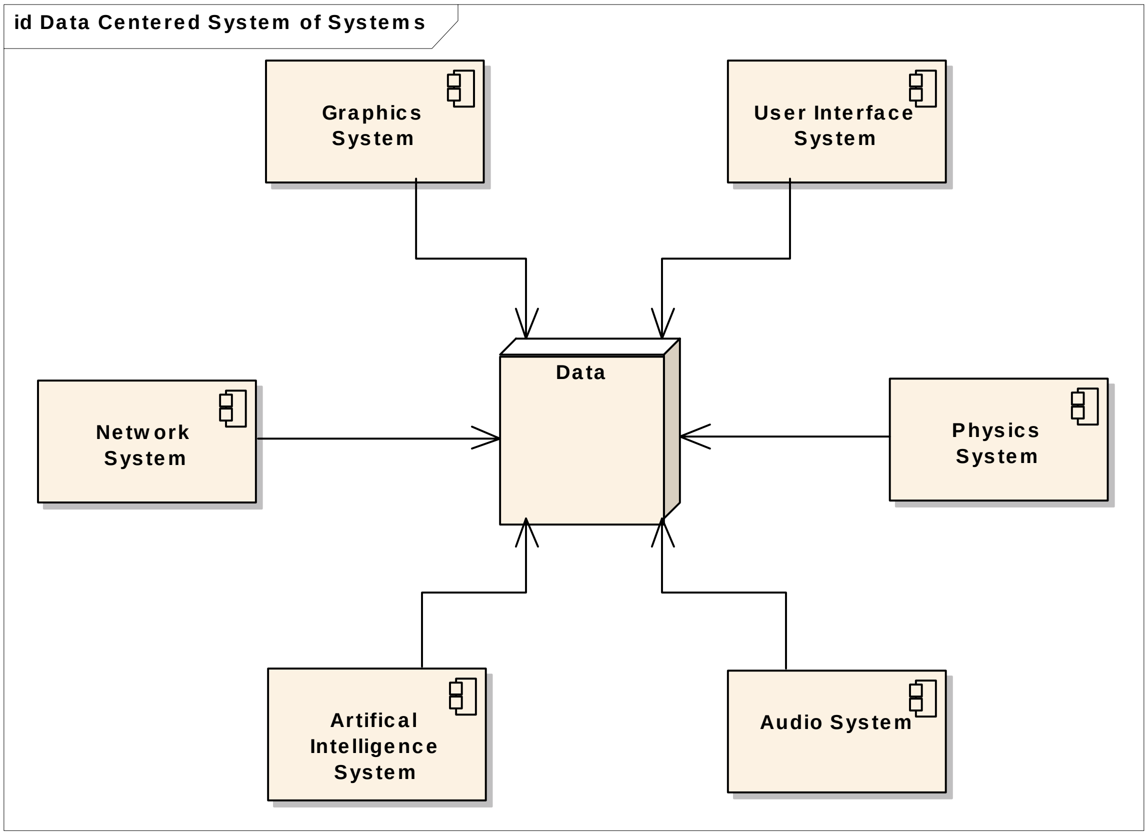


Figure 20 - Data Centered System of Systems

One nice feature of the design shown above is the minimization on dependencies. Sub-systems no longer depend on each other, they can only work with data and by working with the same data they are working with each other. Such decoupling should mean that any sub-system could potentially be replaced or modified without breaking any of the other components.

The concept presented in Figure 19 is definitely an interesting approach but it has one fatal flaw that any gamer would immediately notice – speed. Games are expected to run at very fast speeds, any thing less and the product would be summarily dismissed as a failure. The above design would suggest that each system processes on the whole of the data. In an era where the data content of a single game can span multiple CDs, this is obviously not a feasible approach.

This brings us to the second major design decision – selective data processing. Taking a page from existing game development knowledge, we know the mathematically complex and time consuming graphics system doesn't need to process all data objects in the game, only the objects in the player's immediate area. In fact just about every sub-system could benefit from some sort of spatial data organization or scene management. By moving from a simple data store to a complete data management system we can move back closer to the performance of existing game architectures (See Figure 20).

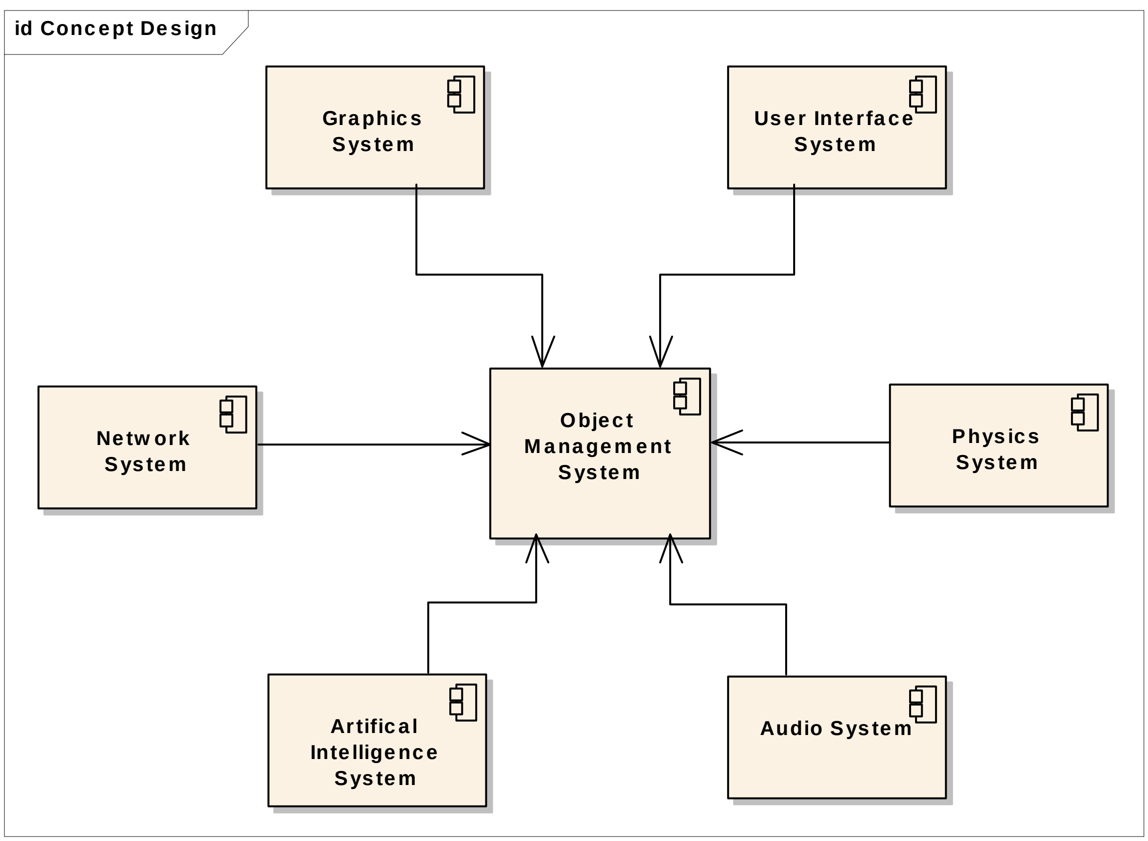


Figure 21- Intelligent Data System Centered System of Systems

4.2 Architecture – System Communication

As shown in the analysis phase, system communication can be performed a variety of different ways. Continuing with the system of systems idea, each domain-specific component working with the object management component is actually an independent system (see Figure 21 below). Since a complete system is composed of only two components and single connection, a direct connection or function calls is acceptable. Allowing direct communications is actually preferable considering the performance constraints that exist in the games domain.

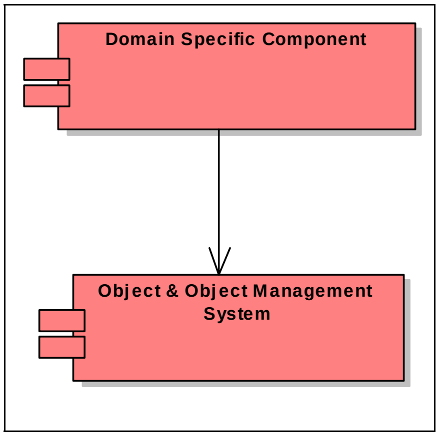


Figure 22 – System Defined as a Domain-specific Component & the Object Component

Allowing direct interface communication between a domain-specific component and the object management system fortunately doesn't have much impact in terms of flexibility and expandability. All domain-specific components require virtually the same types of interactions with the data. They only require lists of objects to operate on, and the ability to read and write to those objects. As long as the design supports those kinds of interactions, the system should remain easily modifiable and expandable.

4.3 Architecture – Synchronization

Software architects might immediately notice that this architecture doesn't support much in the way of component synchronization. There is no direct communication between domain-specific systems so there is no immediate way for one domain-specific component to tell another that it has modified data the other was using. In some application domains this could be a very serious problem, but keep in mind this architecture is for games. If for a single tick an object gets drawn even though the AI system determined that it was killed, a player isn't likely to even notice let alone care.

Synchronization does exist, but it is performed at the system level rather than the object level. Unlike architectures where synchronization exists at the object level, it is no longer enough to "tick" each object in the relative scene and trust that the object will be drawn, act out its behavior, make sound, etc. Now each system must execute on the data in turn. A master system must tell a component to operate on all the relevant data and then signal the next component to do the same. (see Figure 22 below).

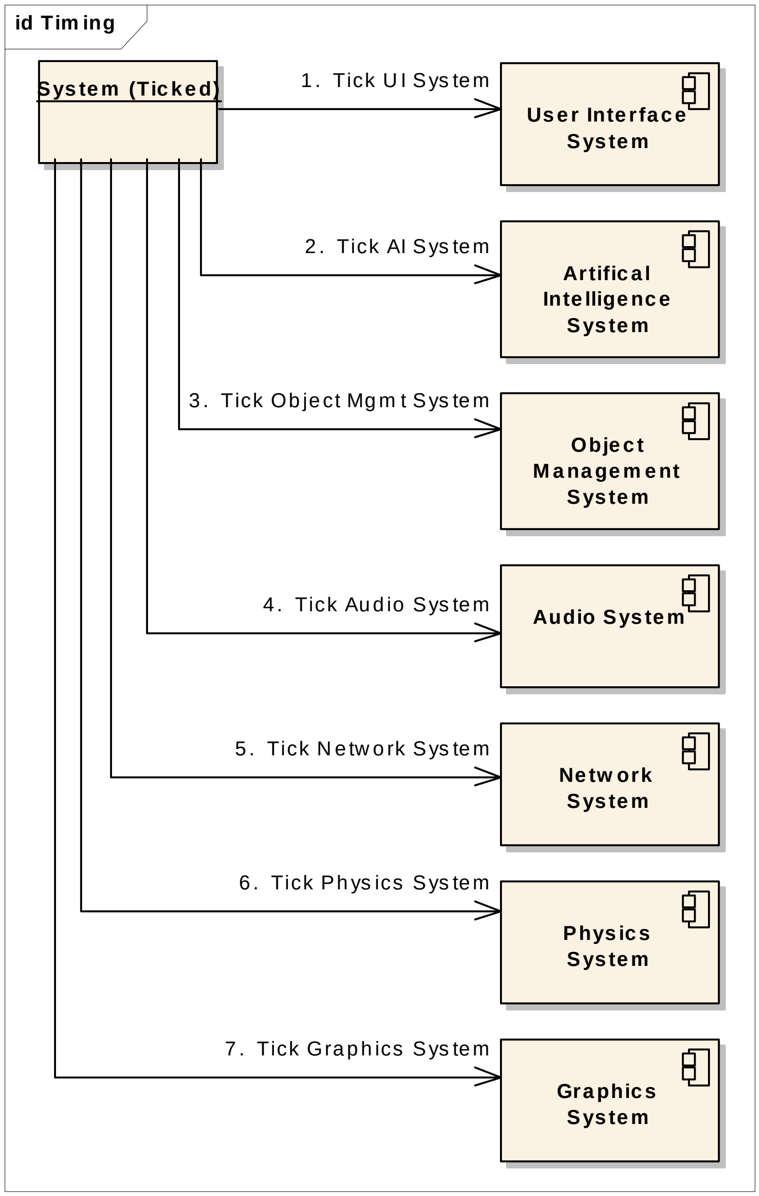


Figure 23 - Ticking the Game System of Systems

4.4 Architecture – Distributed Synchronization

It is important to note that the architecture assumes each component is operating on the same computer. This method of synchronization is not plausible if the domain-specific systems resided on different platforms. This does not mean, however, that distributed games cannot be developed using this architecture. In fact creating a networked game is a simple expansion of adding a networking component to the local system.

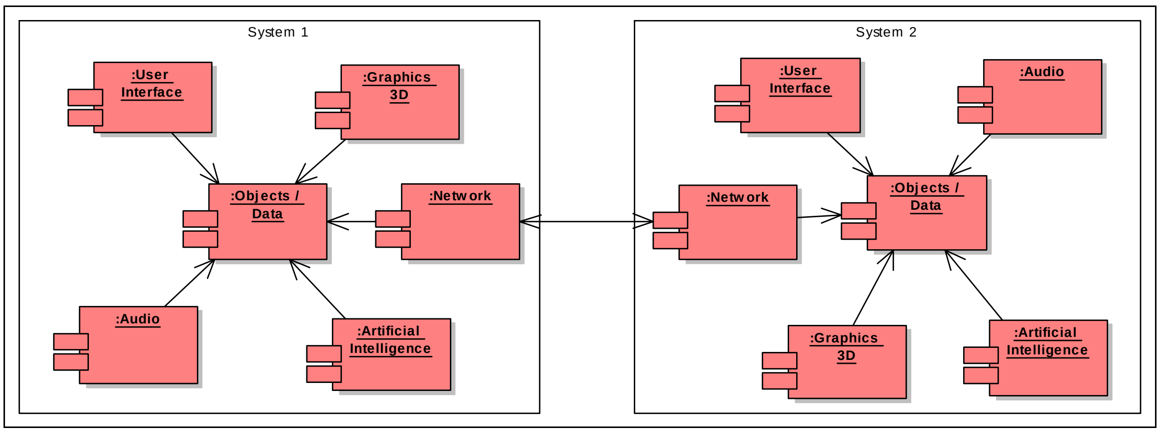


Figure 24 – Example Peer to Peer Networked Game

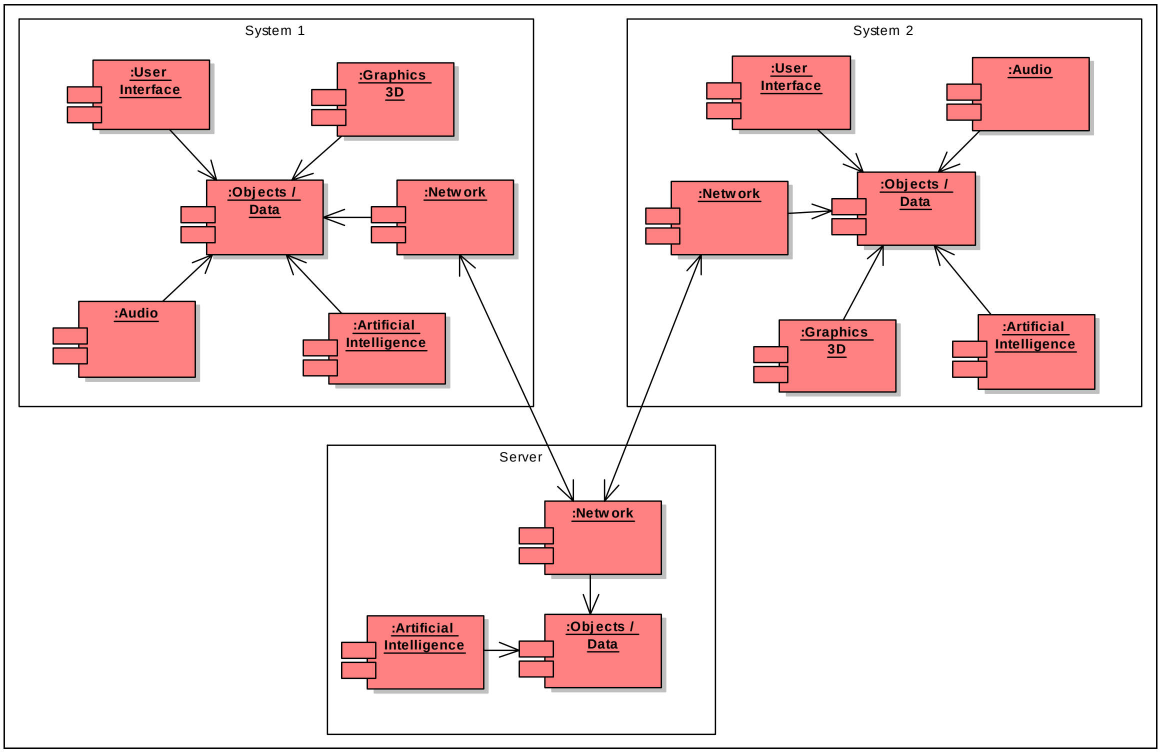


Figure 25 -Example Client Server Networked Game

As you can see in Figures 23 & 24 above, the architecture is capable of supporting the most common networking models. Game developers are free to create their own method of game synchronization. You'll notice that the server system in diagram 24 above has not only a network component, but an AI system as well. This was added because games often use estimation logic in the server to keep clients reasonable synchronized due to the fact that different clients have different quality of network connections.

4.5 Architectural Features / Architectural Requirements

The goal of the proposed architecture was designed to meet the requirements stated earlier in Chapter One. While at this stage of the thesis it has not been proven that the proposed architecture will meet all of the requirements, it does look promising. Actually validating the architecture will be presented in later chapters.

4.5.1 Support for COTS-Based Development

The proposed architecture seems to support COTS-based development very well. Functionality is separated and integration is reduced to a simple logical interface. As long as the COTS component can request objects to process, and is capable of operating on the object data, game systems should have little difficulty integrating external systems.

4.5.2 Better Knowledge Localization

At this point it isn't immediately discernable whether the architecture supports better knowledge localization than other approaches. In one sense it does because the only cross component communication is that of requesting objects to operate on, thus removing the need for game developers to learn complex domain-specific APIs. On the other hand, the object system must support domain-specific data in order for the components to operate properly. We shall see a little later on that this concern can be mitigated in the design phase.

4.5.3 System Flexibility / Modifiability

The architecture appears to be flexible. The data-centered topology allows for any type of system to operate on the data, so seemingly any type of game could potentially be created. It is the developer's choice of components and their functionality that determines the type of game being produced. In later chapters this thesis will attempt to more solidly prove that the proposed architecture supports this requirement.

4.5.4 System Expandability / Maintainability

The figures above represent potential game systems comprised of several subsystems. While there is nothing wrong with the potential game design, the diagrams don't show one of the architecture's greatest strength – expandability. The figures above show one artificial intelligence system executing in a game, but there is no reason there can't be more.   Consider the possibility of having one AI system that determines unit strategy, while another performs path-finding from one location to another across a map (see Figure 25 below).

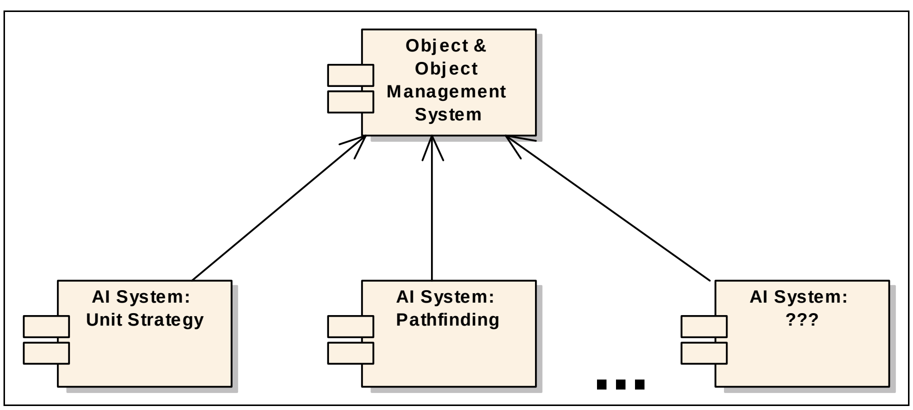


Figure 26- Potential Design using many AI Systems

The architecture readily supports the ability for game designers to use any type and number of subsystems they choose. This also promotes COTS development and re-use, since developers can easily re-use some of the more general subsystems, like path-finding AI, across multiple games.

4.6 A Simple Design

In order to verify the architecture is even feasible, a very simple design will be created. The design is not intended to be the official starting point for games to begin development from. It is simply meant to ensure that it is possible to create a game system using the proposed architecture. Future research will include building better designs using this architecture, but for this thesis simplicity is the only requirement.

4.6.1 Potential Design: System Communication / Interaction

The architecture defines the topology of any design components, so the fist step is to determine how the individual systems will collaborate via the object management system to form a cohesive game system. Using the proposed data-centered approach there are two kinds of interaction that are of interest. First is the interaction to attach an external system to the object management system. It should be generic enough that any number and type of component should be able to attach in a similar fashion. The second important interaction is the actual reads and writes that take place between the object system and an external system. The method of interaction must be generic enough that all subsystems can use it, and flexible enough to support the different kinds of interactions domain-specific components will need.

4.6.2 Potential Design Cont.: Attaching Systems at Compile Time

The ability to attach any type and any number of systems to the object management system is critical to the architectural requirements for flexibility and expandability. Because the details of the design are only interesting from an architectural feasibility standpoint, the simplest design was taken and systems will connect to the object component via semi-standardized interfaces (see Figures 26 & 27 below).

The approach below is definitely not the best but it does work. Domain-specific systems require the object management system to implement a specific interface. The domain-specific system will then communicate with the object system via that interface. It is system expandable at compile time by having the object system implement a new domain-specific interface and having the game system of systems attach the new domain-specific system. This design is not too bad if the interface the object system is required to implement is kept simple, which it will be.

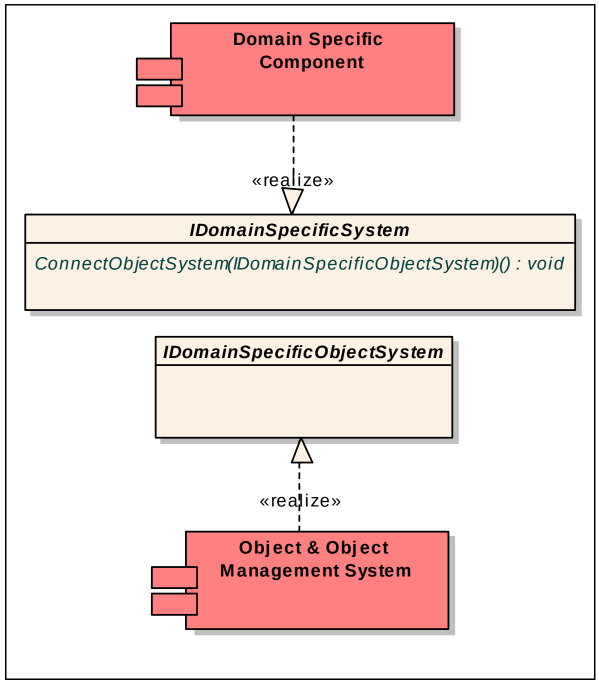


Figure 27 – Interfaces Required to Connect Domain-specific Component to the Object Management Component

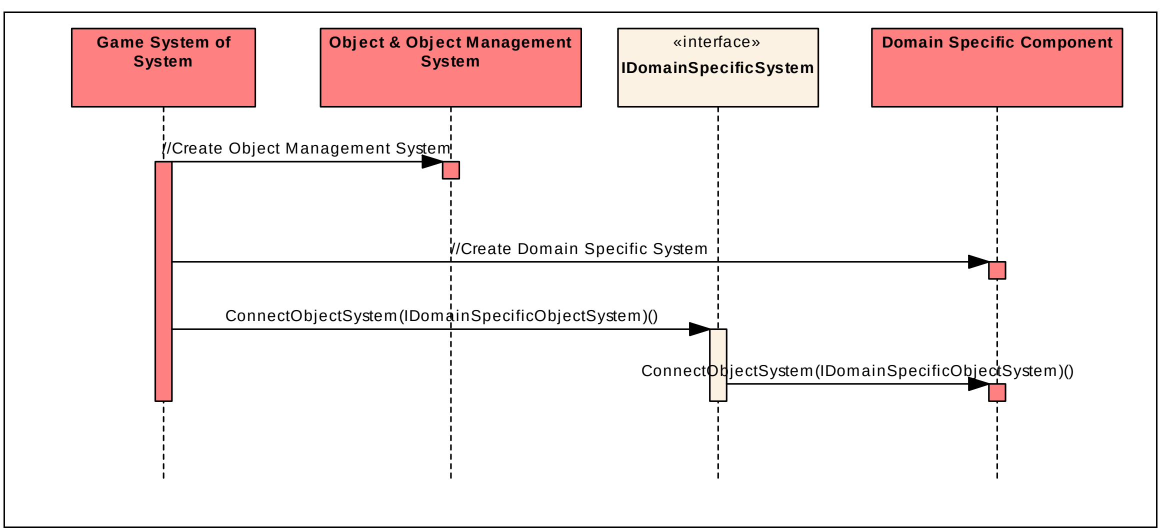


Figure 28 – Example Sequence of Connecting a Domain-specific Component to the Object Management Component

4.6.3 Potential Design Cont.: System Communication

Continuing with the design presented above we need a design that provides a simple and generic way for the domain-specific systems to interact with the object management system. Fortunately the interactions required is simply one of getting data objects for domain-specific processing. I've found that a view/object-list interface provides generic enough access, and is flexible enough to meet the data access needs of the domain sub-systems (see Figures 28 & 29 below).

Essentially each domain-specific system needs to request object lists or iterators of objects to process. The view provides constraints and a context for the object list. For example, a graphics engine requires lists objects that should be drawn. In order to do this the graphics engine might receive a view that provides context stating these objects should take up the whole screen, and then provides the list of visible objects to draw. It might also receive a small view that states to draw the contained objects in the upper left hand corner, and provides a list of GUI objects to draw.

An AI system, on the other hand, might only require a single view that allows the AI system access to all the objects within 100-meter radius of the player, or perhaps a simple list of computer controlled creatures. So while both the graphics and AI systems require different lists of objects, the view / object-list approach is flexible enough to meet the needs of both.

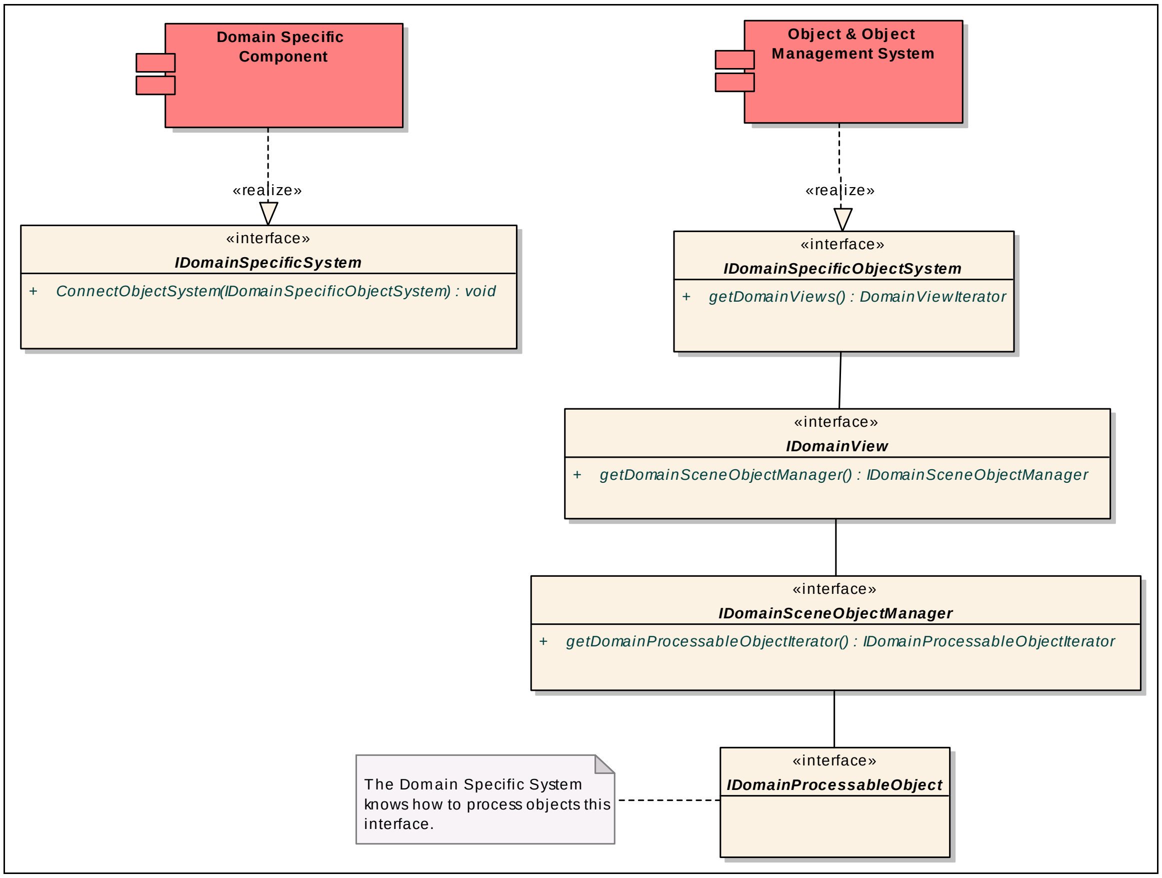


Figure 29 – Interfaces Required for Domain-specific System To Request Objects to Process

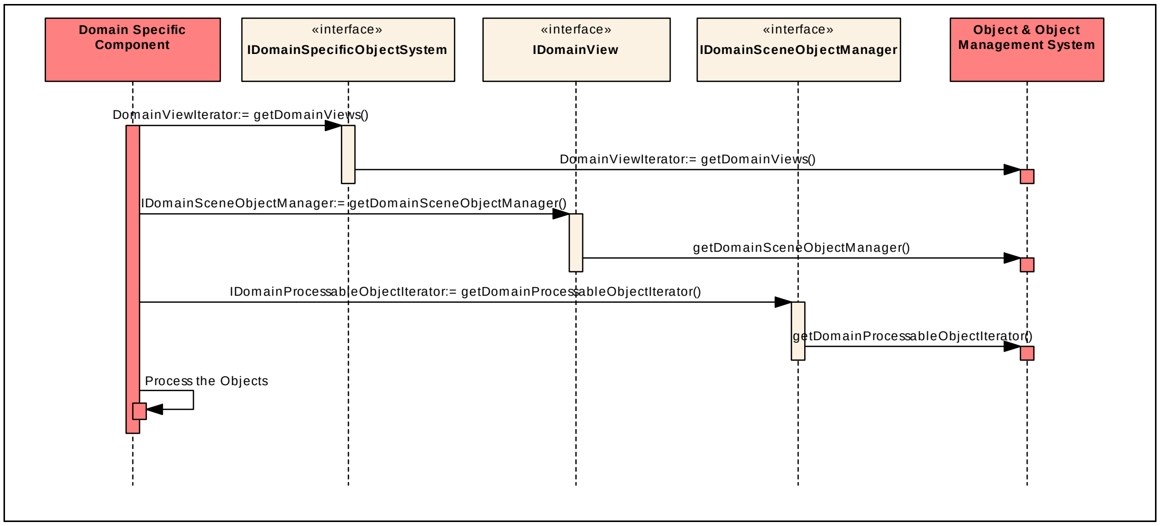


Figure 30 – Example Sequence of a Domain-specific System Requesting Objects to Process

4.6.4 Potential Design Cont.: Observer Pattern to Achieve Localization of Domain Knowledge

One of the initial architectural requirements is to support domain knowledge localization. For example, the graphics system contains a great deal of domain data like mesh and animation structures that is directly related to the game objects in the object management system. The designer of the object management system and even the game specific objects should not need to know about those domain-specific details. A game developer should care that a game object is “attacking”, not necessarily that a specific graphics engine, with specific class objects is being used to represent the attack visually.

One possible solution to this problem is the observer design pattern (Bass et al). If objects in the object management system had the generic capability to attach and retrieve observer objects, domain-specific systems could attach domain-specific data for processing without the object system needing to understand the data. Figures 30 and 31 below show a simple example of how a simple object can be expanded to contain domain-specific data without the game object creator needing to understand the specific domain. So for example, the graphics engine could attach an object that contains the 3D mesh, a skeleton, material information etc. as an attached object, and the game object need never know it contains graphics specific information.

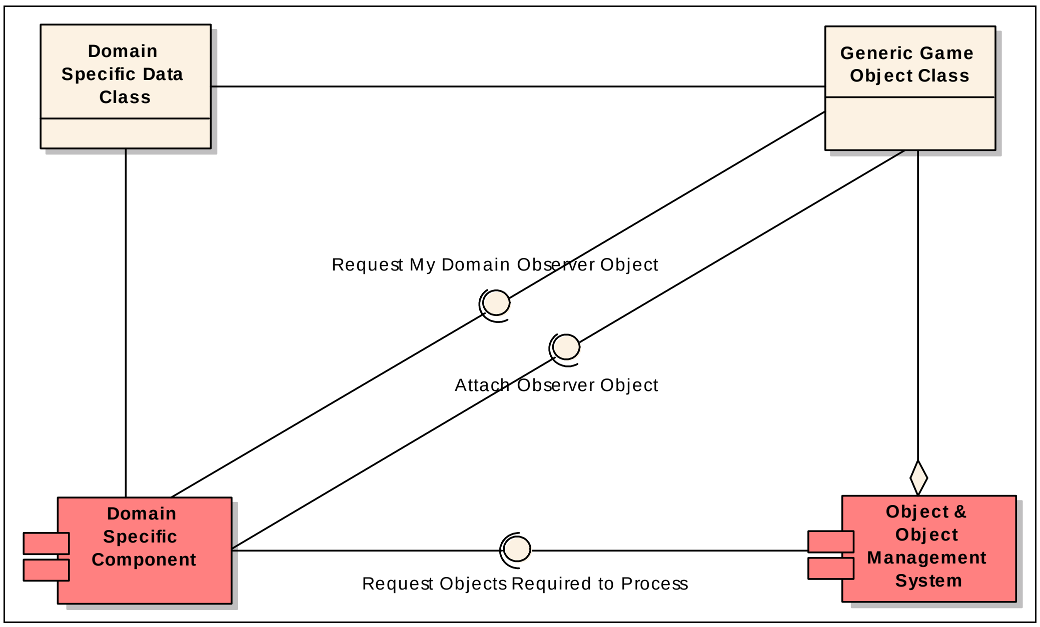


Figure 31-Potential Design using a Domain Observer Object

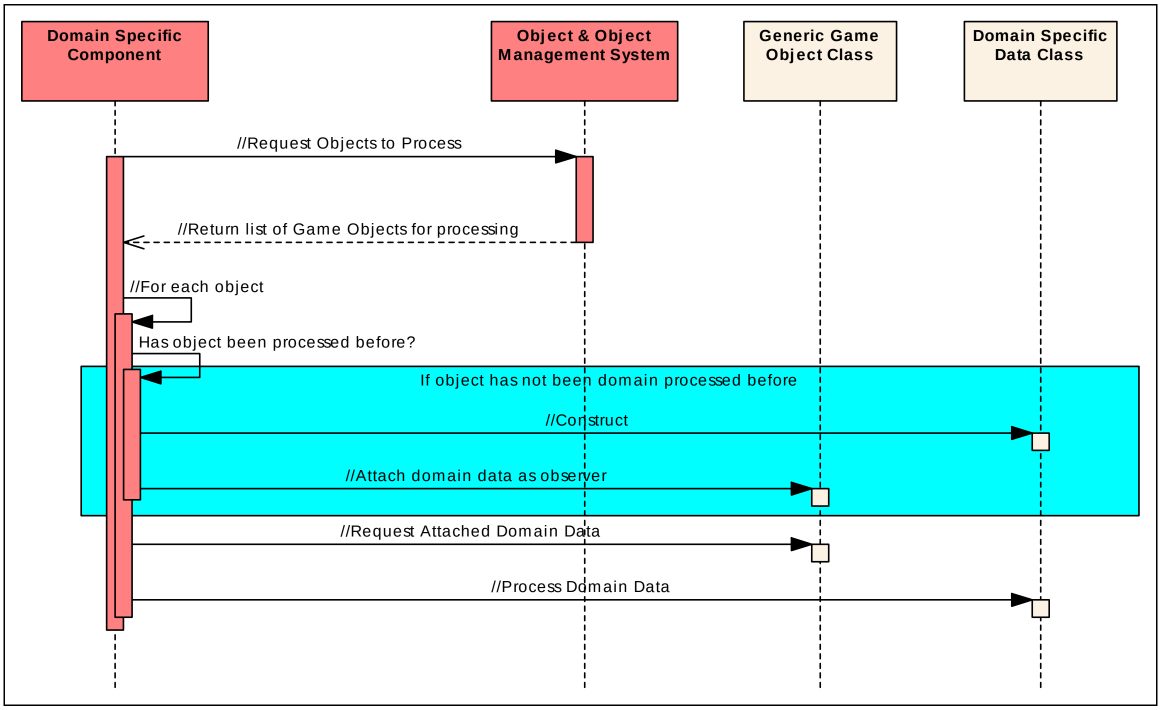


Figure 32-Potential Sequence using a Domain Observer Object

5 ARCHITECTURE VALIDATION

The next step in developing the architecture is to verify to a reasonable degree that the architecture supports the functionality for which it was intended. The first approach is to apply the architecture and take the reference games past the functional level to the design level. This should prove to a fair degree of certainty that the architecture still supports the different game functionality.

The next validation technique used in this thesis is to build a prototype system using the proposed architecture and confirm that the original goals and requirements have been met. Obviously building a commercial quality game like Starcraft™ or Unreal Tournament™ are beyond the scope of this thesis, but building a prototype that offers a subset of functionality can be created in a reasonable amount of time. The prototype system should demonstrate each of the original architectural requirements.

5.1 Taking the Reference Games to the Design Level

5.1.1 Applying the Design

The first step in proving the architecture is sound, is proving the architecture can at least support the functionality it was designed for. Carrying the original game analysis to the design level should show that the games could have been built using this architecture. This step will not, however, show if the architecture would work well for the given game application. The architectural qualities will be left for the prototype to demonstrate.

Before we can carry on to software design, the original analysis artifacts must be reviewed to see how the proposed architecture affects our understanding of the game. Going back to the “Play Starcraft” use case diagram in Figure 9, there is one major problem that needs to be solved. While it appears to capture the activities a human player can perform, it is still incomplete for trying to understand how games really play. The reason is the timing model for games is very different than the typical software application.

Most literature proposes use-cases to capture the interactions between actors external to the system and the system being developed. Games are slightly different, however, in that the player does not initiate all forms of interactions. For example, if a player starts a game of Starcraft™ and never enters another command, the game will still play. The computer AI will process strategies, units will move and behave, and ultimately the game will continue without the player. By bending the rules slightly and treating the clock as an actor, the transactional use-case approach should still be sufficient for capturing the functional requirements in our design.

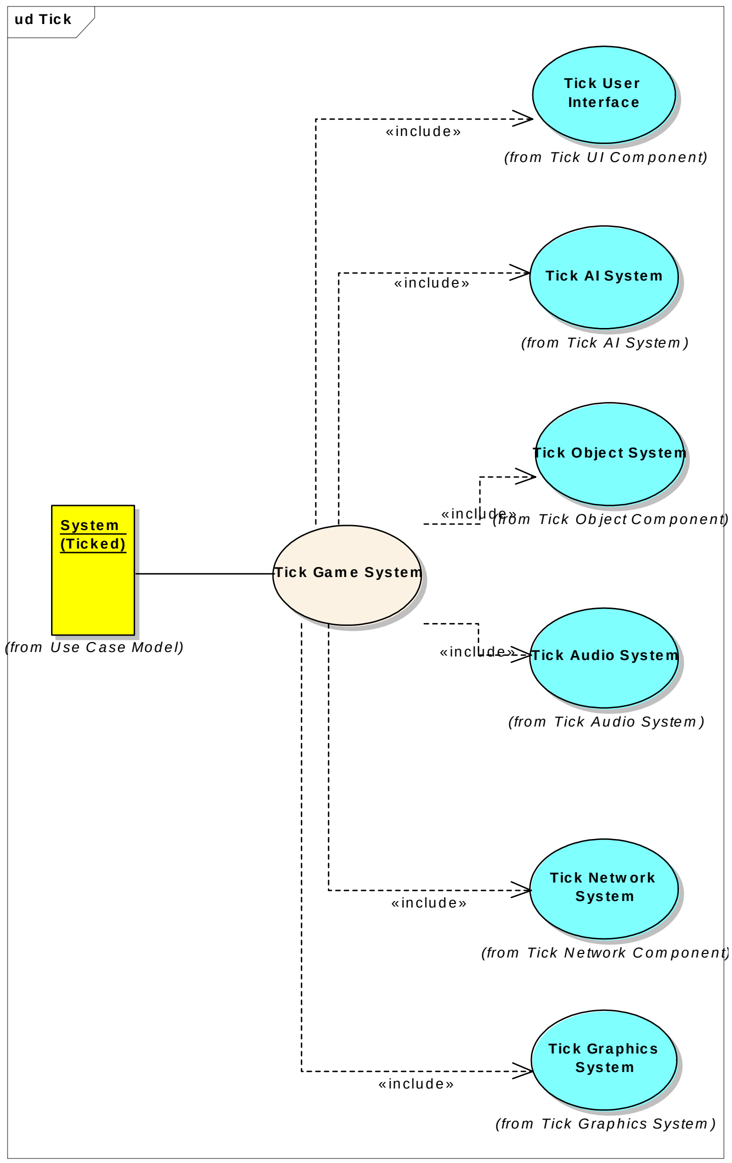


Figure 33 - Tick Game System Use Case

By looking at the original use-case list in Figure 9 in terms of how they would break out in terms of the timing use-cases in Figure 32, we can start to understand how the logical subsystems might implement the game functionality. From here we can begin to break down the use case and assign portions of it to the various sub-systems. Figure 33 below shows a possible use-case breakdown of the “Tick Graphics System” use-case for the game Starcraft™.

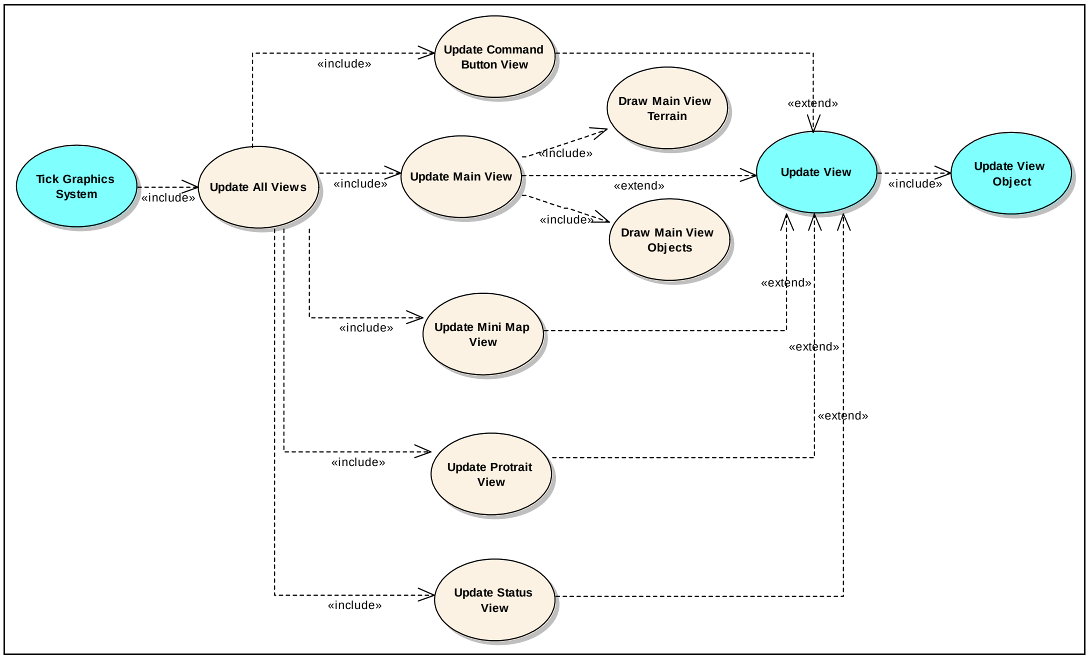


Figure 34 – Tick Graphics System

Selected use cases are then further expanded similarly to what was done during the analysis phase, only this time the simple design is used. Use cases are driven down to the system interactions, which are then further driven down into the actual interfaces involved (see Figures 34 and 35 below). At the end of this we have not only validated that the analyzed games could be like be built on the proposed architecture, but we have further defined the interfaces which will be useful for the prototype effort. For the detailed designs of Starcraft™ and Unreal Tournament™ see appendix A.

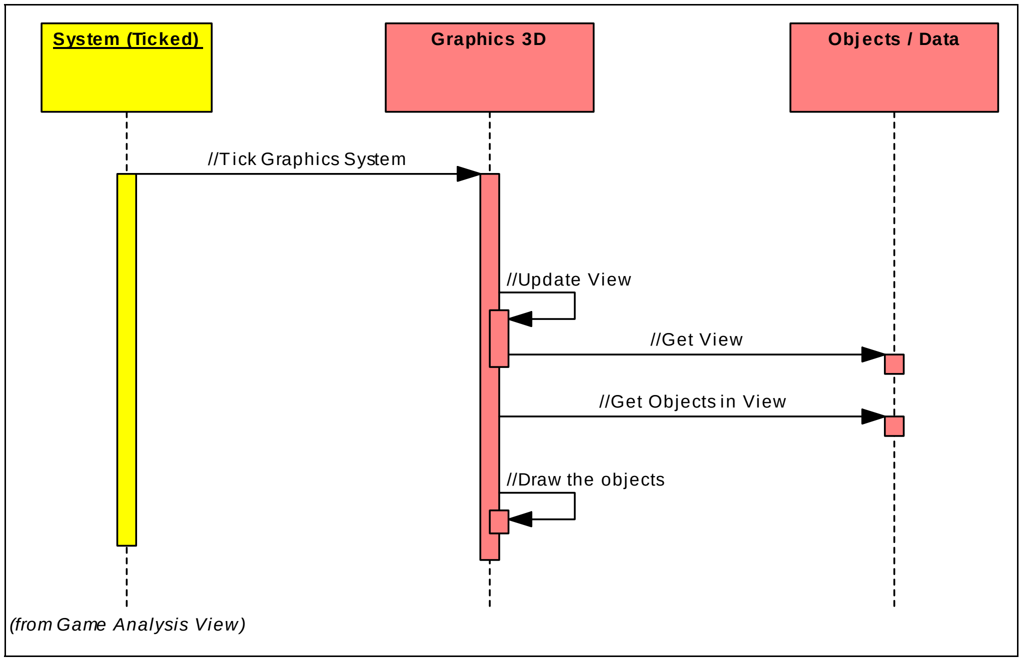


Figure 35 – Update View Component Sequence

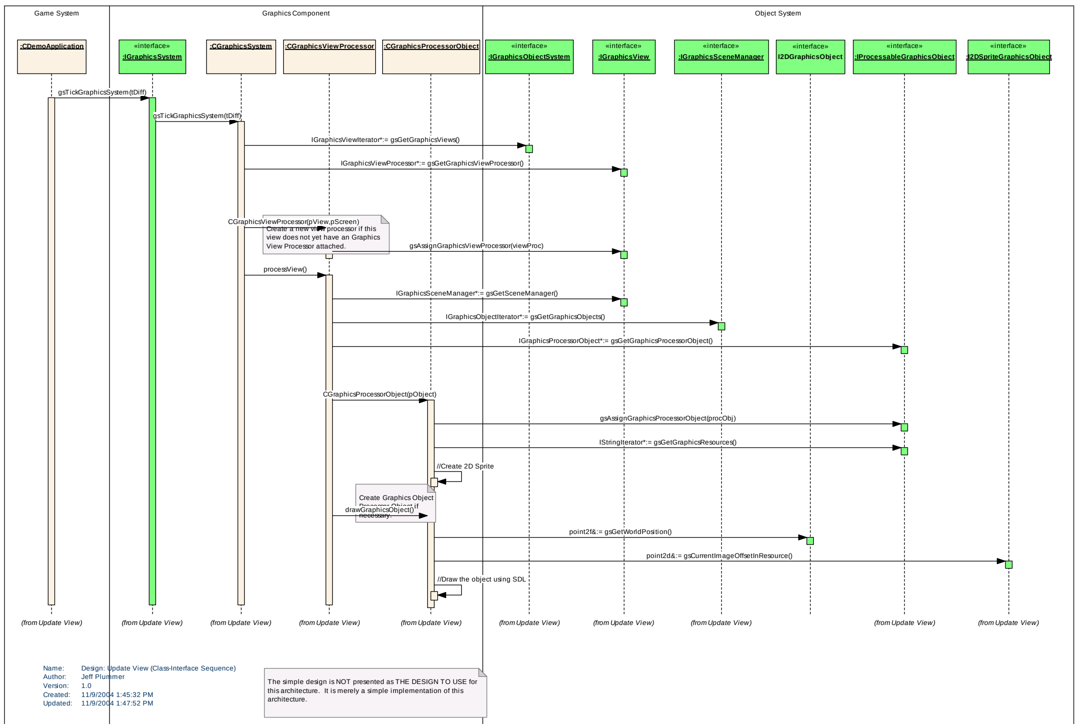


Figure 36 – Update View – Classes and Interfaces

5.1.2 Evaluating the results of applying the design

Before we analyze the results of this exercise it's important to truly understand the goals. Moving the selected games down to a design level was performed to verify that these types of games could be built using this architecture. The resulting artifacts do not give any insight as to how well the architecture fits the games domain. The artifacts also are design dependent, so the level of complexity in these representations is more a reflection of the quality (or lack thereof) of the design, and not the architecture.

Overall it would appear that the architecture can support the two selected games, and therefore arguably supports many types of games. Using the proposed architecture it was possible to design the kinds of functionality required for both games. The idea of individual systems operating on the same data was a bit of a paradigm shift from what is commonly seen in game development literature, but the shift was not so large to make it a difficult transition.

5.2 Developing a Prototype

In an industry where changing people's perceptions of software engineering is so difficult, a paper analysis of the architecture is not likely to change anyone's development habits. A tangible prototype that can demonstrate the architectural qualities in a game-like application is far more likely to have an impact. A prototype will also more concretely prove the quality attributes this architecture purports to have.

5.2.1 Prototype High Level Design

An effective prototype for this thesis needs to meet certain criteria. First the prototype must have game-like functionality. It should demonstrate some of the same kinds of capabilities that exist in games. Second it should demonstrate all of the architectural requirements stated in Chapter One of this thesis. Lastly, even though performance was not one of the architectural requirements, the prototype should execute at speeds reasonable to games. An application that meets such criteria should be able to answer a great many of the questions likely to arise from people familiar with game development.

5.2.1.1 Component Selection

The first task in developing the prototype is deciding which systems to model and build. In order to best demonstrate the architecture's support of our defined requirements, most notably flexibility and expandability, only a few domains will be developed. AI and graphics seem the logical choice and should offer ample opportunity to flex and expand.

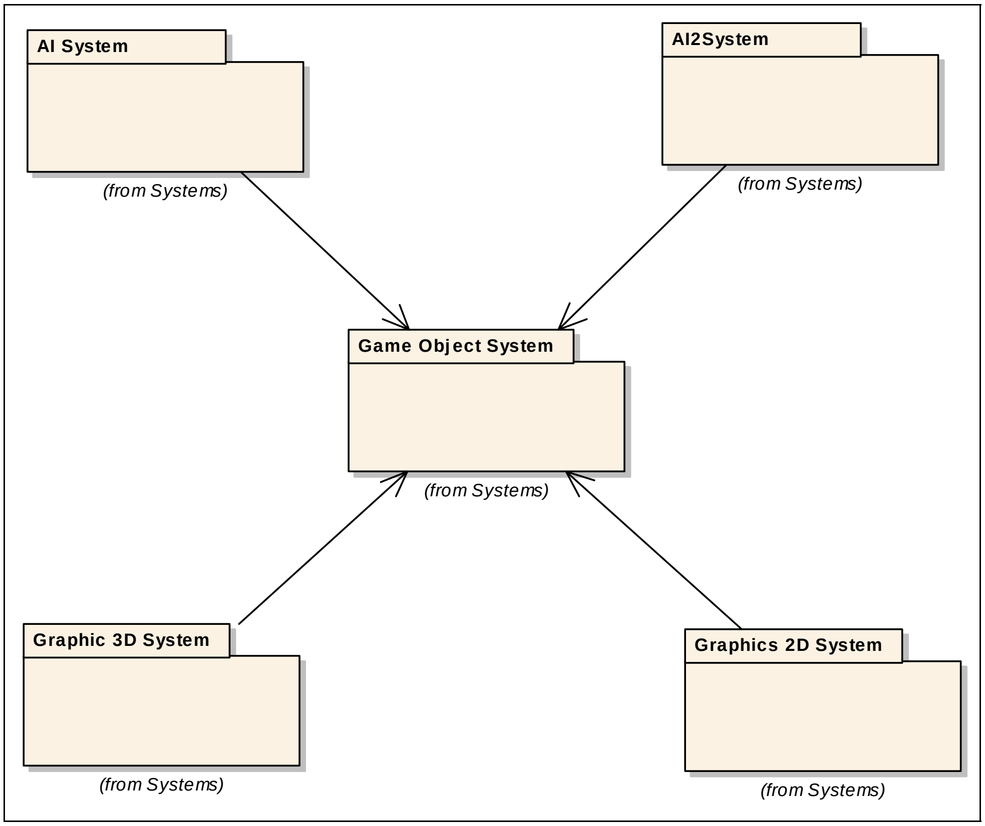


Figure 37 – Prototype Subsystems

Figure 36 above shows the logical systems that will be built for the prototype. The prototype should show flexibility in the way the "game" can be assembled using any combination of these components. It should also demonstrate expandability because moving from a 2D graphics system to a 3D graphics system is a logical upgrade.

<u>Game Object System</u> - This component acts as the data store that all other systems will interact with. It is also responsible for organizing the list of objects the domain systems will operate on.

AI System - This is an extremely trivial intelligence system that will tell objects to move around.

AI2System - This is another trivial intelligence system that tells objects to rotate.

Graphics 2D System - A 2D graphics system that renders sprite objects.

Graphics 3D System - A 3D graphics system that renders 3D objects.

5.2.1.2 The Object Data

The next step is to identify the object data that each system uses to operate on. Figure 37 below shows the object data required for this prototype. This example design also shows how a single data set can be re-used. For example, when the 2D graphics system requests an object position as a point2d (structure of two integers) the object can simple simply return integer typecasts of the x & y aspects of its point3f (structure of 3 floats) location. So in essence when the AI system modifies the position data, it’s modifying the position data that all the components use.

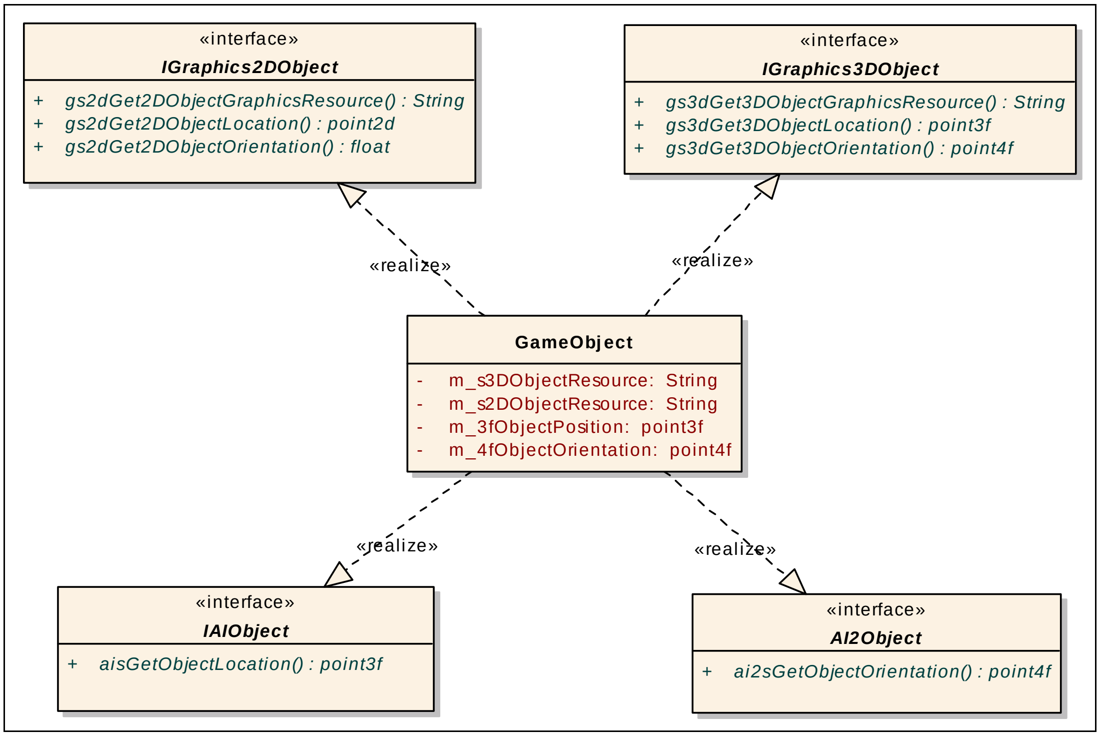


Figure 38 – Analysis of Object Data Required

While this prototype diagram suggests that the object implements interfaces from the various systems, that is a design choice not an architectural requirement. Other designs may use other (possibly better) methods of interacting with the data in the object. Our purpose here is simply to ensure that we understand what data the attaching systems will manipulate.

5.2.2 Prototype Detailed Design

The prototype system will follow the design proposed earlier in Chapter 4. We will soon see it is not the best possible design, but it is simple to understand and adequate for our uses. Components will request views (a view is really just a list of objects as well as some context information), and then process the objects in that view. So for example when a graphics component requests a view, the object system would provide a view that contains the list of likely visible objects.

The design also uses domain-specific observer objects to be attached to the data objects. This allows the domain-specific system to attach domain data to the object without the object component requiring any kind of special understanding of the domain data. As stated before, this design feature was added to provide for knowledge localization.

5.2.2.1 Component Interfaces

The simple design uses interfaces to facilitate communication between the domain-specific system and the object management system. Each domain-specific component will present two kinds of interfaces. One set of interfaces the domain-specific system will implement and present to the game maker / object management system. At its simplest, these interfaces are ONLY for connecting the object system to the domain-specific system, thus keeping the complexities of the domain hidden entirely from the developer. The other set of interfaces are to allow the domain-specific component to use the object system. At its simplest, these interfaces are ONLY for requesting views and access to certain object attributes (see Figure 38 below).

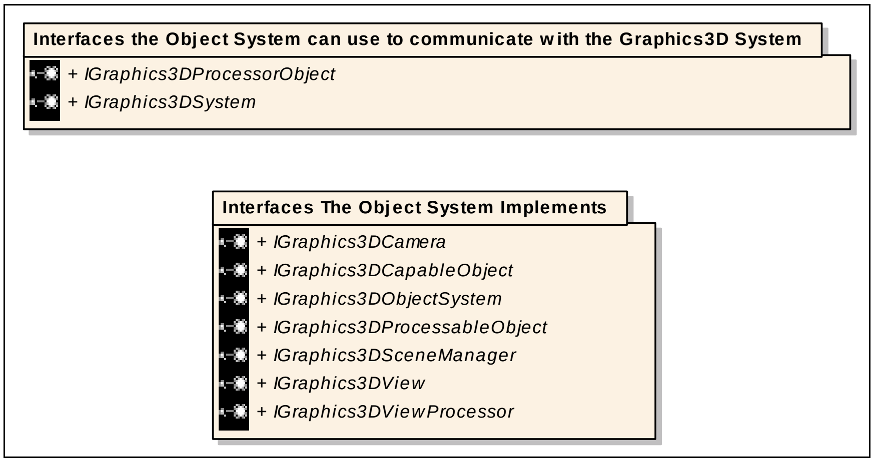


Figure 39 - Example: Graphics3D System Interfaces

In keeping the goal of knowledge localization, the interfaces the domain system presents to the game maker are trivial. In the example below in Figure 39, the IGraphics3Dsystem interface provides mechanisms for the game maker to attach an object system, configure, and “tick” the system. The IGraphics3DprocessorObject and IGraphics3DViewProcessor are the interfaces to allow the domain-specific system to attach observers to a data object and view respectively. Such interfaces could also potentially provide the object and view access to domain-specific functionality, allowing game developer to play with the nuts and bolts of the domain-specific system. They are left empty for this demo because one goal of this demo is to demonstrate that game systems can be assembled without the game developer using any of the domain-specific functionality. Virtually all domain-specific systems will present a nearly identical set of interfaces using this design.

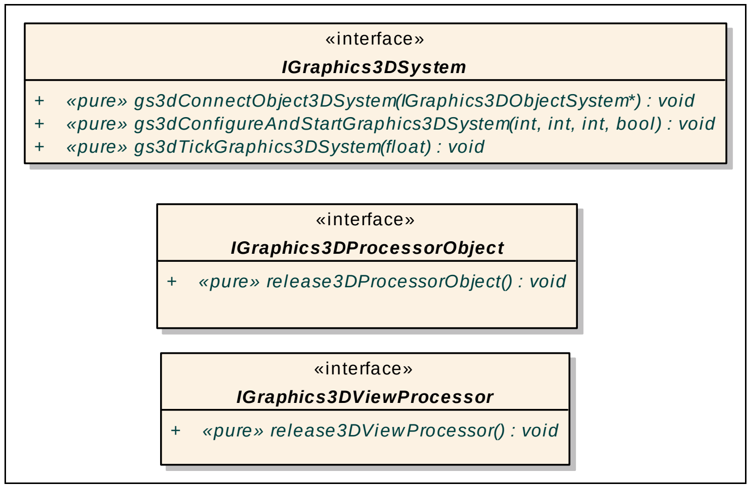


Figure 40 – Interfaces Into the Graphics 3D System

The interfaces the domain-specific system places on the object system to implement are equally trivial. The IGraphics3DObjectSystem interface allows the graphics system to retrieve views. The IGraphics3DView provides access to the objects that should be considered for drawing, as well as context information like the camera and view rectangle. Finally the IGraphics3DProcessableObject interface allows the graphics system to attach an observer, and allows access to the data the graphics system is interested in. And just as before, virtually all domain-specific systems can use a virtually identical set of interfaces using this design.

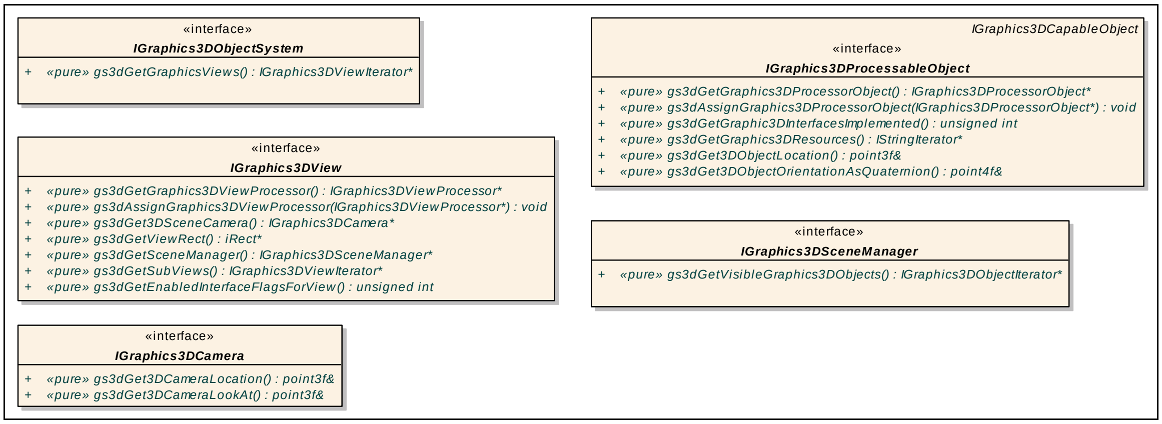


Figure 41 – Interfaces the Object and Object Management System Must Implement in order for the Graphics 3D Component to Use it.

5.2.2.2 Domain-specific System – Object System Interactions

This simple design requires only two types of system interaction. The first occurs at system creation time where the domain-specific system is connected to the object system. The second is the interaction that occurs when you “tick” the domain-specific system. The combination of “ticking” all the domain systems should result in the game system.

5.2.2.2.1 Connecting Domain System to the Object System

The simple design used in this prototype connects the individual systems via interfaces. This extremely simple interaction provides the domain-specific component an interface to use to communicate with the object system. Figure 41 below shows an example of how the system simply passes a reference to the object component to the graphics 3D component. Once the domain-specific component has the interface to the data it can process the data via the “tick” command.

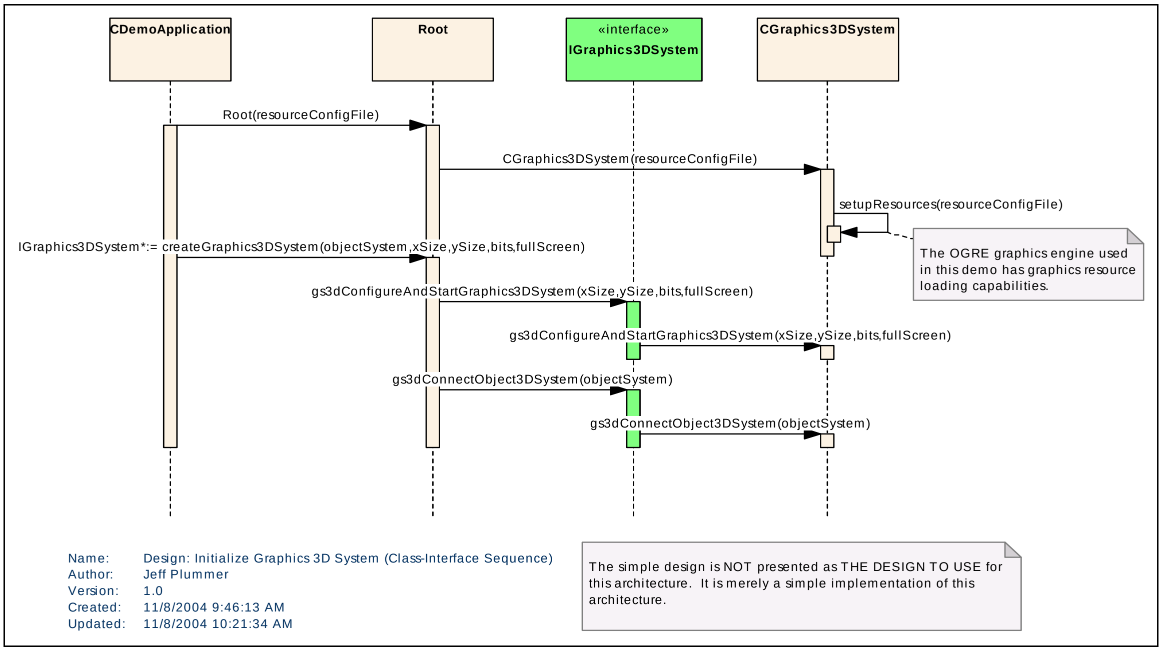


Figure 42 – Connecting the Object Component to the Graphics3D Component

5.2.2.2.2 "Ticking" the Domain-specific System

"Ticking" the domain-specific system is where the real work is done. The system tells the domain-specific component to synchronize and process the data in the object management component. To do this, the domain-specific component will request views and object lists to process, perform domain-specific functionality, and update the data in the object management component. The simple prototype design uses interfaces to perform this and an example can be seen below in Figure 42. See Appendix B for the complete prototype design.

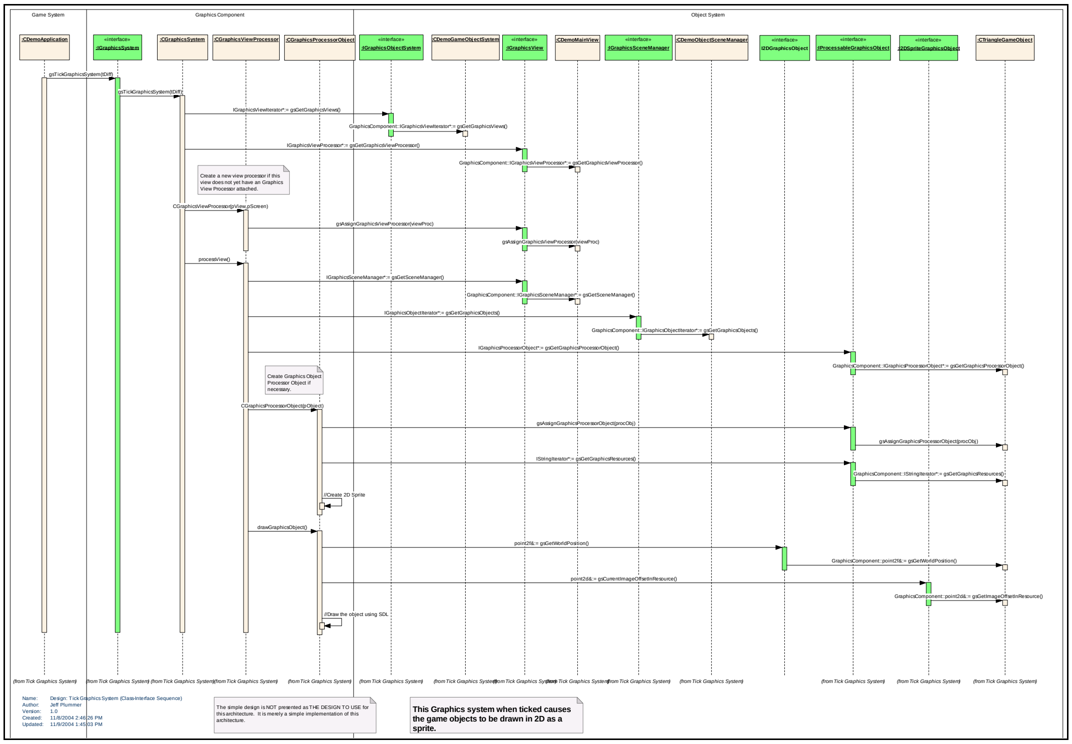


Figure 43 – Prototype Sequence: Tick Graphics2D System

5.2.3 Prototype Evaluation

Designing and building the prototype was a much larger task than originally anticipated, but it was definitely worth the effort. First and foremost it proved the architecture definitely has potential. The prototype not only demonstrated the original quality requirements but it uncovered several issues that had not previously been considered.

In terms of flexibility, the prototype demonstrated the ability to attach and remove AI components quickly and easily, and resulted in notably different "game" behavior. For expandability, the "game" could quickly move from 2D to 3D by attaching a new graphics engine. In terms of domain knowledge localization the game specific objects had no domain knowledge about the components that would use them. The game developer merely had to implement simple data access interfaces, and domain-specific data was hidden as an attached observer object. Lastly, the prototype did prove the architecture supports COTS based development. The domain-specific components were separate by some very simple interfaces and required no understanding of the inner workings of other systems.

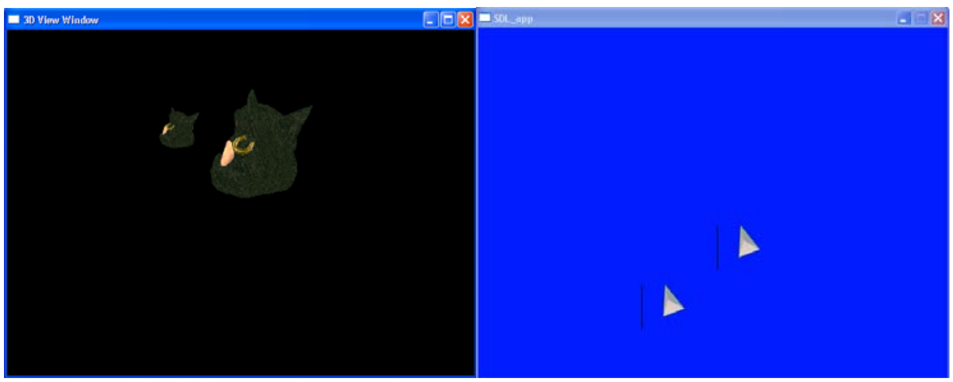


Figure 44 – Screenshot1 from Prototype

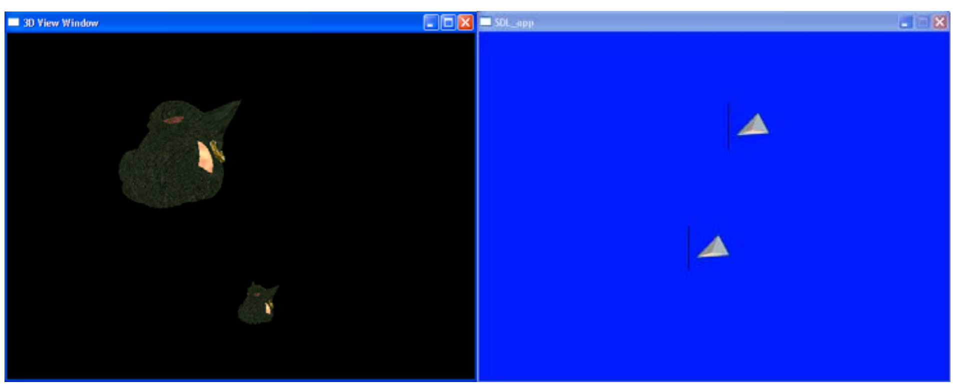


Figure 45 - Screenshot 2 from Prototype

6 RESULTS

As technology advances and consumers demand the latest features, electronic games will be required to continue to grow in terms of size and complexity. In order for development houses to keep costs low, certain realities must be faced. Games can no longer be coded entirely from scratch. The total cost in terms of time and resources will soon reach a point making games an infeasible venture. This thesis has proposed an architectural solution that could help mitigate this problem by moving games to a common COTS architecture. By allowing developers to "assemble" the game framework in a flexible manner from technology components, game makers can spend more of their time focusing on the more game specific aspects.

6.1 Summary

Electronic games are making incredible advances in terms of technology and complexities. Unfortunately almost all-available literature on the subject of games only tends to keep pace with the technology advances, leaving developers to devise their own solutions for managing the complexities. The emerging field of software architecture is an area of research that has been shown to drastically impact the development of large complex systems. By using the knowledge found in software architecture and applying it to the games domain, we can begin to fill the gap that is left by current literature.

In order to design a quality domain-specific architecture, a solid understanding of the games domain needed to be acquired. Such insight was achieved by analyzing existing games using standards software engineering practices. The resulting artifacts presented a quality understanding of the kinds of functionalities and interactions that can occur in modern day games. Then came the task of finding architectural styles that offer a nice fit for those kinds of interactions. The resulting research also sparked a profound interest in the system of systems philosophy. Weighing the pros and cons for each style as applied to the domain resulted in a solid knowledge base for designing an architecture that could meet the needs of the games domain.

The next phase was to actually use all the acquired knowledge and design an architecture for the games domain. The proposed architecture was defined as using a data-centered topology, a direct method invocation for communication, and using a system “tick” for synchronization. The game system emerges as a result of multiple independent systems working on the same data set. A simple design was then created that could be used for the remaining analysis.

After building a simple design it was time to begin analyzing the architecture. The first method of analysis was to carry the selected games for analysis down to the design level. This should verify if the architecture is at least capable of support the analyzed games and by implication capable of supporting many other types of games. In order to determine how well the architecture would support games the simple design was used in a prototype system. The prototype was then used to demonstrate the quality attributes of the architecture.

6.2 Conclusions – Meeting The Architectural Requirements

The proposed architecture definitely shows promise for use in the games domain. It appears to support the functionality required in a diverse set of electronic games, and it appears to support them quite well. The architecture allows for a great deal of flexibility and expandability by supporting any number of a wide variety of domain-specific systems. The architecture also supports the COTS based development approach, and the easy integration of those components. All in all the architecture seems to offer a great deal of benefits over the more ad-hoc, tightly coupled designs used today.

6.2.1 Support COTS-Based Development

The proposed architecture promotes COTS-Based development by eliminating domain-specific component communication. Domain-specific components can only communicate with the object component, and together form an independent system that exists completely independent of any other system.

This architectural requirement was verified during the development and assembly of the prototype. During development, components required only a simple object system to create a fully demonstrable and testable system. During prototype assembly, the prototype components can be added and removed at will, demonstrating the complete independence of the individual components. The domain-specific components also integrate in a near identical fashion simplifying their integration.

6.2.2 Better Knowledge Localization

The proposed architecture also supports the ability to localize domain knowledge away from the game developer. By eliminating, or at least greatly reducing, the amount of domain knowledge a game developer must understand in order to use a game component properly, we are effectively giving the developer more time to focus on the game specific aspects of the code.

While such a feature is not immediately inherent in the architecture, it is possible to add this at the design level. This thesis expanded the architecture to a design that used the observer design pattern to attach domain-specific data to a game object. In the prototype, the game specific objects have very little domain-specific data. For example, the only 3D Graphics specific piece of data the object component has is a string that says what 3D graphics resource to use. All the underlying data that is needed to render that resource is attached as an observer, and is effectively hidden from the game developer.

6.2.3 Flexibility / Modifiability

Flexibility and modifiability are important to allow developers to re-use components in a wide variety of games. So by investing money in an expensive piece of domain-specific technology, the developer has not severely limited the kinds of games he/she can make. The proposed architecture is flexible enough to allow developers to mix and match components allowing them to assemble almost any possible game.

This architectural requirement was demonstrated by both the reference games and the prototype. Both very different reference games were able to be designed using the proposed architecture, strongly suggesting the architecture can support a wide variety of games. The prototype also demonstrated flexibility in that attaching different domain-specific components resulted in a variety of “game-like” applications.

6.2.4 Expandability / Maintainability

Expandability and maintainability are important in keeping development time and costs down, and should ultimately result in a better quality upgrade. Iterative game incarnations are most often technology upgrades with minor tweaks in game play, and should not require complete redesigns. The proposed architecture supports this capability by keeping the domain-specific components independent of each other, allowing technologists to upgrade each system without breaking the other functionality.

The prototype demonstrated this requirement in a few different ways. First, as stated in meeting COTS-based development, the systems are truly independent of each other allowing technologists to modify components without breaking others. Second, the prototype shows expandability by demonstrating technology upgrades by swapping in entirely different systems. The prototype application was able to make the technology upgrade from a 2D graphics system to a 3D graphics system by simply attaching a different component. Lastly expandability is supported by the architecture in much the same way it supports flexibility - systems can be expanded by simply adding a new component.

6.2.5 The Performance Concern

Although Performance was not an official requirement of this architecture because the assumption that most the performance issues reside inside the components, it is definitely something game developers would be concerned about. Reviewing the prototype and the simple design, it appears as though this architecture has very little impact compared to the more monolithic designs presented earlier.

First the architecture allows for direct interface invocation, not requiring any message-handling overhead (although the architecture does not preclude the use of using messages as the method of communication). Second, the architecture doesn’t create much in the way of extra communication. For example, whether an object calls a graphics library or the graphics systems requests an object to draw, the number of interactions is the same. The exception comes from the fact that there is no direct communication between the domain-specific components. When components need to communicate, they must write data to the object management system, and wait for the response in the next system tick cycle.

The prototype seems to support the notion that performance is not significantly affected by the architecture. In quick comparisons between the samples that came with the Ogre™ graphics engine, and the prototype there were no significant performance differences. Although more detailed profiling would be required to prove how much the architecture affects performance; that is beyond the scope of this thesis.

6.3 Important Considerations

Developers considering this architecture should read and understand some of the important considerations that will affect development. These are a few items of wisdom that were found during the work on this thesis.

6.3.1 Design is Critical

One important fact when using this architecture is that the architecture “supports” many of the quality attributes. The design plays a large role in determining whether those quality attributes are part of the system. One such example is the quality of knowledge localization. This quality wasn’t realized until the design phase where the observer pattern came into play.

The design can also negate some of the implied quality attributes of the architecture. For example, the architecture also “supports” easy component integration by limiting the communication between the domain-specific component and the object management component to simply requesting objects, and read/writing data to those objects. The prototype design, where the object component is forced to implement interfaces for each attached component, makes component integration quite tedious. So while it is possible to design complex game systems with the proposed quality attributes using this architecture, it left to the designer to ensure those attributes are realized in the system.

6.3.2 Central Object Management System = VERY different

This architecture uses a very different topology than the designs of today. The current trend seems to be that every system has its own object management system – e.g. graphics & 3D sound engines each have their own way of organizing objects. This makes the libraries easy to use, but it duplicates functionality.

One of the goals of this thesis is to promote COTS based development where specialists can design the best and most optimized components. By centralizing object management into one area, it means specialists can build the best management algorithms, whether BSP trees, Oct trees, etc. without being concerned with some of the domain specialties. It also means people writing the domain-specific components, like sound, need not concern themselves with complex scene management.

Many game developers may take issue with this approach making arguments that items like a graphics engine may have highly optimized scene management specialized for that particular graphics engine, and that a 3rd party scene manager would impact performance. The thing to realize is that this is a design concern, not an architectural concern. The architecture merely states that the object management component will provide a domain-specific component with objects to process. There is no restriction saying that a specific graphics component can't recommend a specific optimized scene management system to use. By placing it in a central location, however, that same scene management system is available for the other systems to use.

The architecture also doesn't state that there is only one scene management system within the object management component. The object component may have multiple scene managers that the different systems can use. For example one scene manager may be designed to provide a list of objects in the player's view that the graphics engine will use. Another scene manager could exist that is designed to provide a list of objects within a specific radius of the player that is used by the sound and AI components.

6.3.3 Think about the Data

When designing to this architecture it is important to think about the data that will reside in the shared data store. Part of the benefit of this design is that the data you place there is usable by all domain-specific systems. For example, objects in the prototype had location and orientation that was used by both the graphics systems and the AI systems.

Another issue related to data is the data types used. Since the domain-specific components may be written by different companies, and so may be expecting slightly different data types. The graphics engine may want "double" precision floating point values for location, while the sound engine may require integers. While this problem is no different than current games using 3rd party libraries, it shows itself in a slightly different manner.

6.4 Future Research

During the course of completing this thesis a great many ideas were left on the drawing board because they were beyond the scope of this thesis. They are captured here as ideas for future research, and represent many of the interesting problems that remain to be answered.

6.4.1 Can this Architecture Work for Massively Multiplayer Online Games

Massively multiplayer games represent the next big advancement in electronic entertainment. The enormous number of distributed players and objects present some very interesting problems that were not considered in the design of this architecture. It will be interesting to see if this architecture can scale across multiple servers, with thousands of players, all existing in a persistent world.

6.4.2 Design: Domain-specific Component Connection to the Object Management Component

The simple design used for the prototype, forcing the object management component to implement interfaces, is very weak. While forcing objects to implement data access interfaces may be necessary to maintain performance, attaching components and requesting object lists don’t have the same restrictions. A better design would allow domain-specific components to easily attach to the object management system, and request objects to process.

6.4.3 Design: No More Interfaces to Access Object Data (If performance allows)

While function calls to retrieve the data is probably the fastest method to access object data this architecture can support, there may exist more generic methods that don't greatly affect performance. For example, if a simple query language methodology could allow domain-specific components to access object data without a significant cost in speed, the ability to add and change components is made significantly easier.

6.4.4 Architecture Inside the Components

While the focus of this thesis was designing the architecture at the inter-component level, architecting the components themselves is still relatively uncharted territory. It would be in interesting assignment to research the domains and see if a common architecture could be created for the specific components. If no such architecture exists, which is likely due to the diversity of the domains, then work should begin designing reference architectures for each of the domains.

6.4.5 What is messaging overhead for independent component style

The independent components and system of system architectural styles were rejected in this thesis because it was thought the messaging overhead were to high for game systems. It would be an interesting experiment to see how much that overhead would affect performance. If messaging does not cause a significant drop in performance many other architectural possibilities are made available.

6.4.6 The Architectural Tradeoff Analysis Method

An important piece of work is left undone in this thesis, and that is the architectural tradeoff analysis method (ATAM). Due to time restrictions not all quality attributes could be analyzed. It would be an extremely worthwhile endeavor to truly analyze this architecture more completely, looking at those quality attributes that were not tested.

Works Cited

“3 Million Lines of Code.” EdGames. Sept. 13 2004. <http://edweb.sdsu.edu/courses/edtec670/edgames/2002/12/3-million-lines-of-code.htm>.

“3D Engines Database: Unreal Engine 3.” *DevMaster.net*. Sept 14. 2004. <http://www.devmaster.net/engines/engine_details.php?id=25>.

“A $30 Billion Dollar Industry.” Aug. 2003. < http://www.xboxcity.com/console/NewsDetail.asp?NewsID=1422&fc=0 >.

“A Brief History of the FPS.” Aug. 2003. < http://www.3dactionplanet.com/features/editorials/fpshistory1/>.

Adolph, Steve. “Reuse and Staying in Business.” *Gamasutra*. 12 Dec. 1999. Sept 12. 2004. <http://www.gamasutra.com/features/19991213/adolph_02.htm>.

Alves, Carina. João Bosco Pinto Filho and Jaelson Castro. “Analysing the Tradeoffs Among Requirements, Architectures and COTS Components.” Centro de Informática, Universidade Federal de Pernambuco Recife, Pernambuco. Sept. 5 2004. <http://www.cs.ucl.ac.uk/staff/C.Alves/WER01_COTS.pdf>.

Bass, Len. Paul Clements, and Rick Kazman. *Software Architecture in Practice.* Addison-Wesley, 1998.

Busto, Roberto Del. “Games and Simulations.” Aug. 2003. < http://coe.sdsu.edu/eet/Articles/gamessims/index.htm >.

Calvert, David. “Software Architectural Styles.” 3 June 1996. Aug. 16 2004. <http://hebb.cis.uoguelph.ca/~dave/27320/new/architec.html>.

"Definition: System of Systems." *The Free Dictionary.com*. Oct. 7 2004. <http://encyclopedia.thefreedictionary.com/System%20of%20systems>.

"Domain-specific Software Architectures." Aug. 2003. < http://sunset.usc.edu/classes/cs578_2003/13-Domain-Specific%20Software%20Architectures%20(DSSA).pdf >.

Duffy, R. "Software Architecture." Sept. 12 2004. <http://members.aol.com/rduffy4187/report.html>.

E. Berard. *Essays in Object-Oriented Software Engineering*. Prentice Hall, 1992.

Fristrom, Jamie. "Manager in a Strange Land: Most Projects Suck." *Gamasutra*. 17 Oct. 2003. Sept. 12 2004. <http://www.gamasutra.com/features/20031017/fristrom_01.shtml>.

"History of Arcade Games." Aug. 2003. < http://www.hut.fi/~eye/videogames/arcade.html>.

"How to Make a COTS Project Fail." Aug. 2003. < http://www.versaterm.com/topic_list/topic16.htm>.

Nilson, Roslyn; Kogut, Paul; & Jackelen, George. *Component Provider's and Tool Developer's Handbook Central Archive for Reusable Defense Software (CARDS)*. STARS Informal Technical Report STARS-VC-B017/001/00. Unisys Corporation, March 1994.

Rollings, Andrew and Dave Morris. *Game Architecture and Design.* The Coriolis Group, 2000.

Sloan, Jason and William Mull. "Doom 3 FAQ." Aug. 2003.

< http://www.newdoom.com/newdoomfaq.php#5>.

"Unreal Tournament History". Oct. 20 2004.

<http://www.unrealtournament.com/general/history.php>.

APPENDIX A -
GAME ANALYSES

TABLE OF CONTENTS – APPENDIX A

SECTION NAME Page

Game Analysis 107

Game Analysis - Use Case and Dynamic View 107

Player 107

System 107

System (Ticked) 108

Modules 109

Game Data 110

Game Logic 110

Technology Modules 110

AI 110

Audio 110

Graphics 110

Network 110

Physics 110

User Interface 110

Starcraft 111

Use Cases 111

Startup 112

Select Multi-Player Game 112

Select Single Player Game 112

SECTION NAME Page

Options Menu 114

End Mission 114

Get Help 115

Get Mission Objective 115

Load Game 115

Modify Options 115

Return To Game 115

Save Game 116

Play Starcraft 117

Attack Unit 117

Change Map Display Area 123

Gather Resources 126

Give unit an order 132

Move to Location 137

Research Technology 142

Select Object 145

Building construct Unit 150

Give Building an order 150

Hold Position 150

Manipulate Object Resources 151

Manipulate Player Resources 151

Modify Doable Commands 151

SECTION NAME Page

Patrol Location 151

Stop Movement........ 151

Unit Construct Building........ 151

Design: Tick Starcraft System........ 153

Tick Starcraft Game System........ 154

Tick AI System........ 155

Tick AI System........ 155

Navigate Map - Pathfinding 157

Attack........ 158

Calculate AI State........ 158

Calculate Next Movement........ 159

Calculate unit action........ 159

Execute Map Watcher 159

Tick Audio System........ 161

Tick Audio System........ 161

Tick Graphics System 164

:IGraphicsObjectSystem........ 164

Update View Object 164

Tick Graphics System........ 166

Update View........ 166

Update Main View........ 171

Draw Main View Objects........ 171

SECTION NAME Page

Draw Main View Terrain ... 171

Update All Views ... 171

Update Command Button View ... 171

Update Mini Map View ... 172

Update Protrait View ... 172

Update Status View ... 172

Tick Network Component ... 173

Broadcast local objects TO server ... 173

Tick Network System ... 173

Update objects FROM server ... 175

Tick Object Component ... 176

Tick Object System / Game Logic ... 176

Update Commander Object ... 178

Update Controlled Object ... 178

Tick UI Component ... 179

Process Keyboard ... 179

Process Mouse ... 179

Tick User Interface ... 179

Unreal Tournament ... 182

Use Cases ... 182

Play Unreal Tournament ... 183

Collect Ammo ... 183

SECTION NAME Page

Collect Health ... 183

Collect Item ... 183

Collect Weapon ... 185

Jump ... 185

Move ... 185

Rotate ... 187

Shoot ... 187

Design: Tick ... 188

System (Ticked) ... 188

Tick Physics Component ... 189

Tick AI System ... 189

Tick Audio Component ... 189

Tick Graphics 3D Component ... 189

Note ... 189

Tick Network Component ... 190

Tick Unreal Tournament Game System ... 190

Tick AI System ... 191

Tick Unreal Tournament Game System ... 191

System (Ticked) ... 191

Note ... 191

Tick AI System ... 191

Tick Player ... 193

SECTION NAME Page

Tick Projectile 193

Tick Audio Component 194

Tick Audio Component 194

Tick Graphics 3D Component 196

Tick Graphics 3D Component 196

Update All Graphical Views 198

Update Character Status Overlay 198

Update GUI Overlays 198

Update Main Play View 198

Update Team Score Overlay 200

Update Weapon/Ammo Overlay 200

Tick Network Component 201

Broadcast Local Objects TO Server 201

Tick Network Component 201

Update Local Objects FROM Server 203

Tick Object Component 204

Tick Object Component 204

Tick Physics Component 207

Calculate Collision Reaction 207

Detect Collisions 207

Tick Physics Component 207

SECTION NAME Page

A - 1.1 *Game Analysis*

A - 1.1.1 Game Analysis - Use Case and Dynamic View

This diagram shows the high level list of artifacts uncovered during the analysis phase of this thesis.

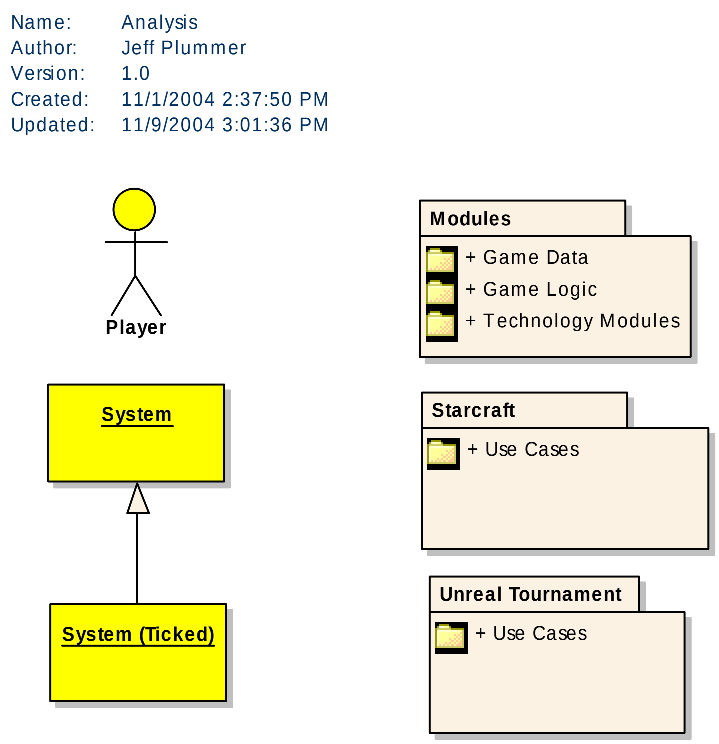


Figure 46 : Analysis

A - 1.1.1.1.1.1.1.1.1 Player

Type: *public* **Actor**
Package: Game Analysis - Use Case and Dynamic View

This actor represents the human player who is playing the game.

A - 1.1.1.1.1.1.1.1.2 System

Type: *public* **Object**

Package: Game Analysis - Use Case and Dynamic View

The system represents "application" portion of the code that will create and tick the components.

A - 1.1.1.1.1.1.1.1.3 System (Ticked)

Type: *public* **Object**
Package: Game Analysis - Use Case and Dynamic View

This actor represents the System but implies the actions occur on a regular or clocked basis.

A - 1.1.1.2 Modules

This package represents the logical modules involved in game development.

This diagram shows all the logical modules involved in game development.

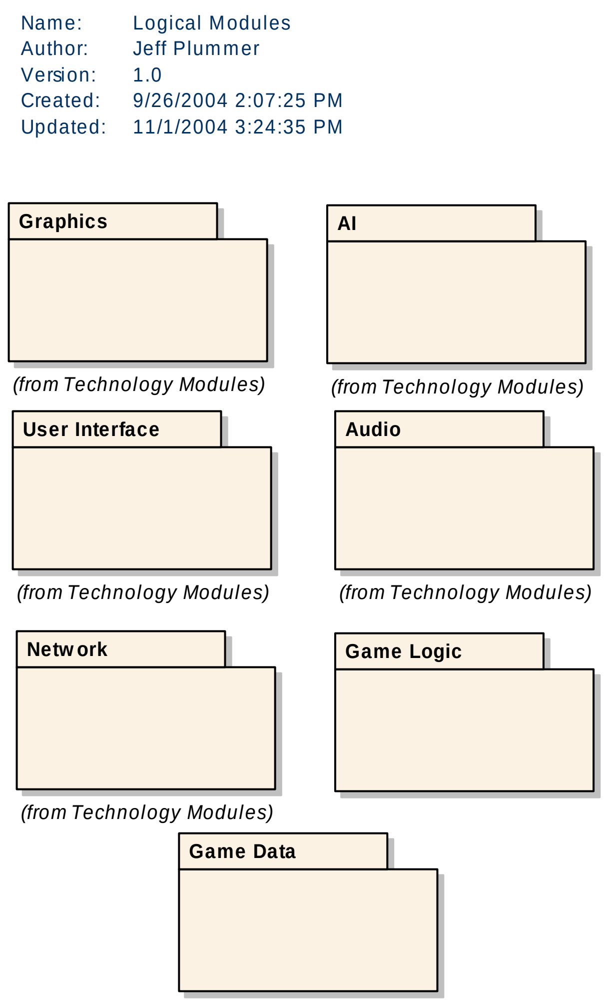


Figure 47 : Logical Modules

A - 1.1.1.2.1 Game Data

This package represents all the game specific data involved in the game.

A - 1.1.1.2.2 Game Logic

This logical module represents the game specific functionality for the system. Game rules, behavior, etc.

A - 1.1.1.2.3 Technology Modules

These packages are the domain-specific logical modules involved in game development.

A - 1.1.1.2.3.1 AI

This logical module represents the artificial intelligence or behavioral functionality required in the game.

A - 1.1.1.2.3.2 Audio

This logical module represents the Audio functionality required in the game.

A - 1.1.1.2.3.3 Graphics

This logical module represents the graphical functionality required in the game.

A - 1.1.1.2.3.4 Network

This logical module represents the network functionality required in the game.

A - 1.1.1.2.3.5 Physics

This logical module represents the physics simulation functionality required in the game.

A - 1.1.1.2.3.6 User Interface

This logical module represents the user interface functionality required in the game.

A - 1.1.1.3 Starcraft

This package represents the analysis and design work performed for the game Starcraft(tm).

A - 1.1.1.3.1 Use Cases

This diagram shows a high level view of the use cases and actors involved in Starcraft(tm).

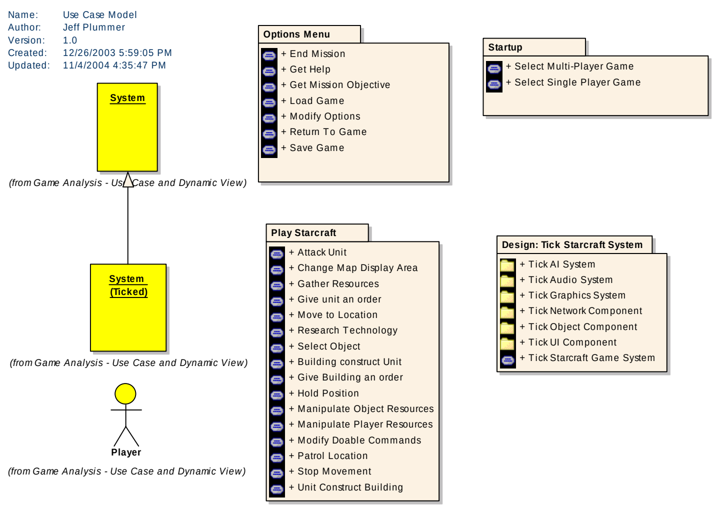


Figure 48 : Use Case Model

A - 1.1.1.1.1.1.1.1.1.1 Startup

This diagram represents the initial options presented to the player when the launch the Starcraft(tm) application.

Name:	Startup
Author:	Jeff Plummer
Version:	1.0
Created:	2/12/2001 12:00:00 AM
Updated:	11/9/2004 2:19:05 PM

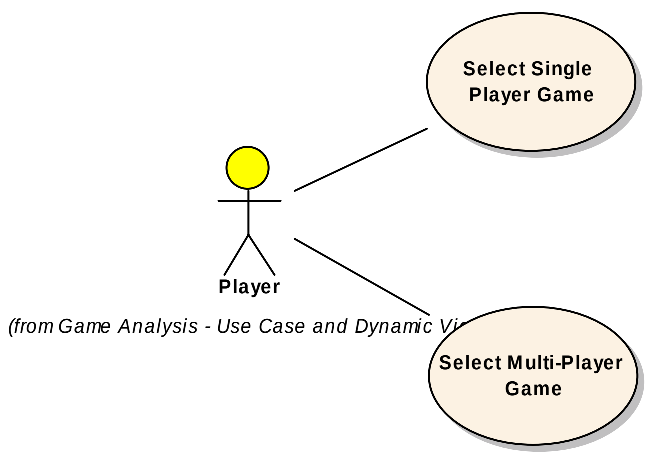


Figure 49 : Startup

A - 1.1.1.3.1.1.1.1.1 <u>Select Multi-Player Game</u>

Type: *public* **<u>UseCase</u>**
Package: Startup

Selecting multiplayer game enables the player to compete against other human players via a network connection or over the internet.

A - 1.1.1.3.1.1.1.1.2 <u>Select Single Player Game</u>

Type: *public* **<u>UseCase</u>**
Package: Startup

Selecting a single player game prepares a game to be played on a single machine against computer controlled opponents.

A - 1.1.1.3.1.2 Options Menu

This diagram shows the options available to the player to choose from in the options menu

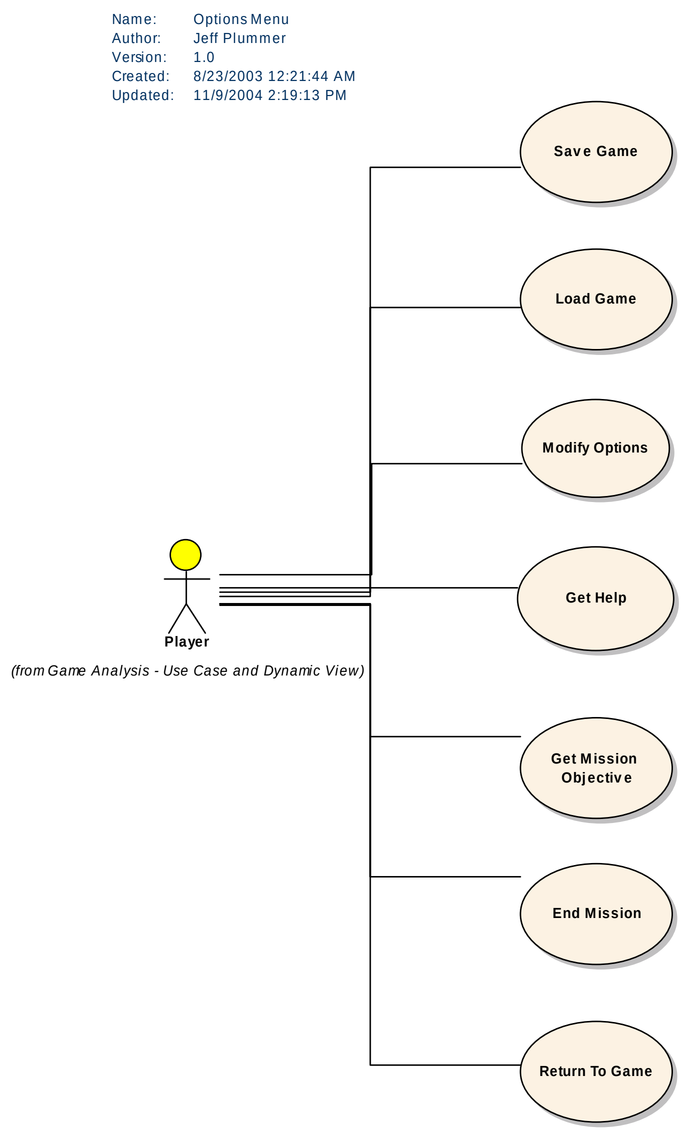


Figure 50 : Options Menu

A - 1.1.1.3.1.2.1.1.1 End Mission

Type: *public* **UseCase**
Package: Options Menu

This use case represents the action to allow the player to end the current mission and quit back to the main startup screen.

A - 1.1.1.3.1.2.1.1.2 Get Help

Type: *public* **UseCase**
Package: Options Menu

Enter the help system.

A - 1.1.1.3.1.2.1.1.3 Get Mission Objective

Type: *public* **UseCase**
Package: Options Menu

This use case represents the action of allowing the player to re-request the list of objectives for the current game level.

A - 1.1.1.3.1.2.1.1.4 Load Game

Type: *public* **UseCase**
Package: Options Menu

This use case reoresents the functionality of loading a game state from a file, allowing the player to continue a game where they last saved.

A - 1.1.1.3.1.2.1.1.5 Modify Options

Type: *public* **UseCase**
Package: Options Menu

A - 1.1.1.3.1.2.1.1.6 Return To Game

Type: *public* **UseCase**
Package: Options Menu

Allows the player to exit the options menu and return to playing the current game.

A - 1.1.1.3.1.2.1.1.7 Save Game

Type: *public* **UseCase**
Package: Options Menu

This use case represents the action of saving the current game state to a file.

A - 1.1.1.3.1.3 Play Starcraft

This diagram represents the actions the player can perform while playing the game.

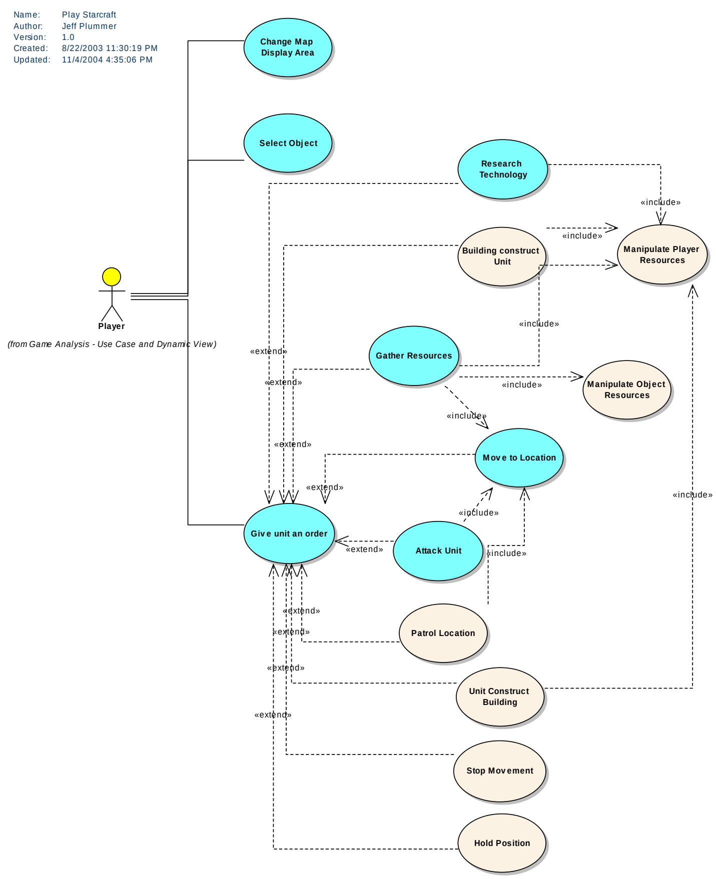


Figure 51 : Play Starcraft

A - 1.1.1.3.1.3.1.1.1 Attack Unit

Type: *public* **UseCase**
Package: Play Starcraft

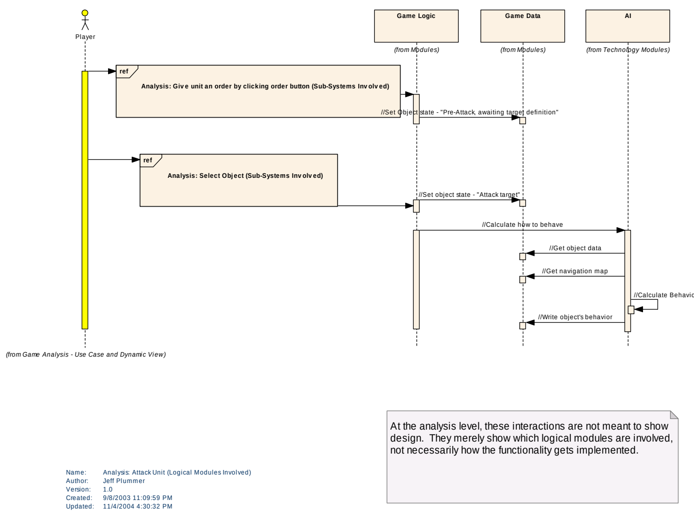
This use case represents the action of a player telling one of his/her units to attack another unit.

Scenarios

Basic {Basic Path}.

1. Player clicks the attack button
2. Player clicks an enemy unit
3. Unit enters attack state, and will move and attack selected enemy unit.

Enemy enters zone of control {Alternate}.
Description:
Without requiring the player to do anything, when an enemy unit enters a unit's zone of control, the unit will attack.



This diagram shows what logical modules are required to perform the "Attack Unit" use-case.

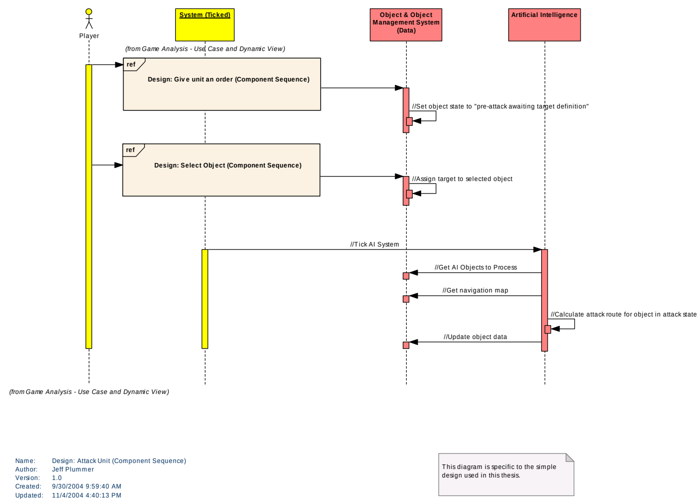
Figure 52 : Analysis: Attack Unit (Logical Modules Involved)

Analysis: Attack Unit (Logical Modules Involved) Messages

ID	Message	From Object	To Object	Notes
1		Player	Analysis: Give unit an order by clicking order button (Sub-Systems	Call the "Analysis: Give unit an order" use case

			Involved)	
2		Analysis: Give unit an order by clicking order button (Sub-Systems Involved)	Game Logic	Represents the beginning of the details specific to this "Give Unit an Order" interaction.
3	//Set Object state - "Pre-Attack, awaiting target definition"	Game Logic	Game Data	Game logic tells the object to prepare for an input selecting the attack target for that unit.
4		Player	Analysis: Select Object (Sub-Systems Involved)	Call the "Analysis: Select Object" use case
5	//Set object state - "Attack target"	Game Logic	Game Data	Tell the object waiting for an attack target, to target the unit that has just been selected.
6		Analysis: Select Object (Sub-Systems Involved)	Game Logic	Represents the beginning of the details specific to this "Select Object" interaction.
7	//Calculate how to behave	Game Logic	AI	The AI logical module will determine how the object should behave - in this case how the object

				will attack.
8	//Get object data	AI	Game Data	The AI functionality requires object data to process like current position, target position, attack range, etc.
9	//Get navigation map	AI	Game Data	The AI functionality requires map navigation data to process. The navigation map is data that says how an object can move from one location to another.
10	//Calculate Behavior	AI	AI	Using the object data and map information the AI logical module will determine what the object should do. It will decide how the object should move (if required) and any attack specific behavior.
11	//Write object's behavior	AI	Game Data	Once the AI functionality has decided what the object will do, the data / state information must be saved inside the object.



This diagram shows the sequence of events at the component level that occur to complete the "Attack Unit" use case.

Figure 53 : Design: Attack Unit (Component Sequence)

Design: Attack Unit (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1		Player	Design: Give unit an order (Component Sequence)	Call the "Give Unit an Order" use case.
2		Design: Give unit an order (Component Sequence)	Object & Object Management System (Data)	Represents the beginning of the order specific interaction details.
3	//Set object state to "pre-	Object & Object Manage	Object & Object Manage	In this design the game logic resides within the game object itself, so the object readies itself for

	attack awaiting target definition"	ment System (Data)	ment System (Data)	receiving the attack target.
4		Player	Design: Select Object (Component Sequence)	Call the "Select Object" use case.
5		Design: Select Object (Component Sequence)	Object & Object Management System (Data)	Represents the beginning of the details specific to this "Select Object" interaction.
6	//Assign target to selected object	Object & Object Management System (Data)	Object & Object Management System (Data)	In this design the game logic resides within the game object itself, the object sets the attack target to the object that has just been selected.
7	//Tick AI System	System (Ticked)	Artificial Intelligence	In this design the AI resides in its own component and will be "ticked" to tell the AI system to operate on a list of objects.
8	//Get AI Objects to Process	Artificial Intelligence	Object & Object Management System (Data)	In this design the AI system will request list(s) of objects to process. The Object management component is responsible for providing the domain-specific component list(s) of relavant objects (i.e. not ALL the objects).
9	//Get navigation map	Artificial Intelligence	Object & Object Manage	The AI system will require some form of traversability map of the object system.

			ment System (Data)	
10	//Calculate attack route for object in attack state	Artificial Intelligence	Artificial Intelligence	Using the object data and map information the AI component will determine what the object should do. It will decide how the object should move (if required) and any attack specific behavior.
11	//Update object data	Artificial Intelligence	Object & Object Management System (Data)	Update the object with data specific to the attack behavior the AI component decided.

A - 1.1.1.3.1.3.1.1.2 Change Map Display Area

Type: *public* **UseCase**
Package: Play Starcraft

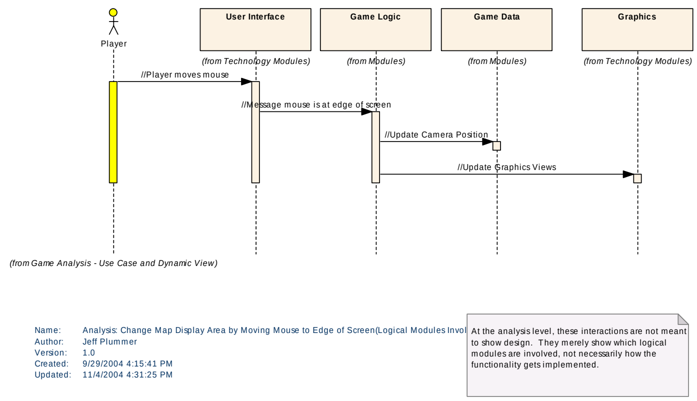
Scroll the main screen showing a different area of the map.

Scenarios

Mouse at edge of display {Basic Path}.
Description:
When the mouse reaches the edge of the visible display, the display will scroll the map smoothly in the direction of that edge.

1. Mouse moves to edge of screen.
2. Move viewable area.
3. Update minimap rectangle.

Click Location on Mini-Map {Alternate}.
Description:
When a user clicks a location on the mini-map, that area becomes the new view area.

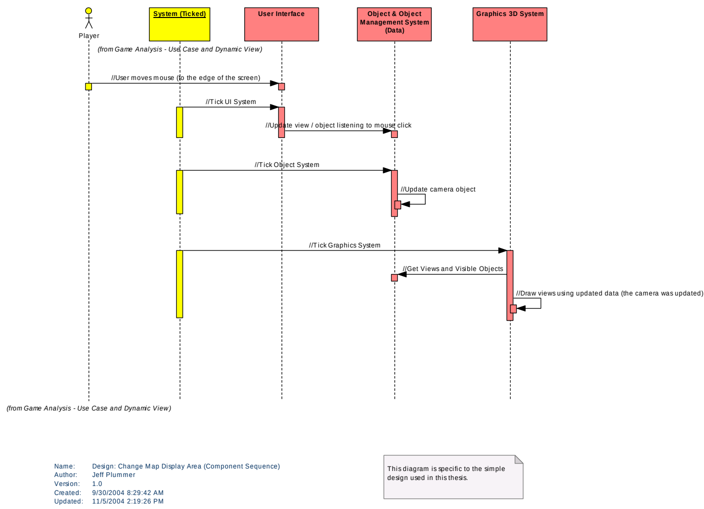


This diagram shows what logical modules are required to perform the "Change Map Display Area" use-case. This is only representative for the sequence where the mouse is moved to the edge of the viewable screen.

Figure 54 : Analysis: Change Map Display Area by Moving Mouse to Edge of Screen(Logical Modules Involved)

Analysis: Change Map Display Area by Moving Mouse to Edge of Screen(Logical Modules Involved) Messages

ID	Message	From Object	To Object	Notes
1	//Player moves mouse	Player	User Interface	The player moves the mouse to the edge of the main view screen.
2	//Message mouse is at edge of screen	User Interface	Game Logic	The game logic needs to know that the mouse has moved to the edge of the screen.
3	//Update Camera Position	Game Logic	Game Data	In order to change the map display area we just change where the "camera" is located.
4	//Update Graphics Views	Game Logic	Graphics	The graphics need to be redrawn using the new "camera" position.



This diagram shows the sequence of events at the component level that occur to complete the "Change Map Display Area" use case.

Figure 55 : Design: Change Map Display Area (Component Sequence)

Design: Change Map Display Area (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//User moves mouse (to the edge of the screen)	Player	User Interface	
2	//Tick UI System	System (Ticked)	User Interface	Ticking the UI system tells the UI Component to grab the status of all UI devices, or atleast update variables based on the UI events that occurred.
3	//Update view / object listening to	User Interface	Object & Object Management	Tell any UI event listening objects in the game object component about the mouse movement.

	mouse click		System (Data)	
4	//Tick Object System	System (Ticked)	Object & Object Manage ment System (Data)	In this simple design the game logic resides in the object system, so ticking the object component is the same as processing all game logic.
5	//Updat e camera object	Object & Object Manage ment System (Data)	Object & Object Manage ment System (Data)	When the game object listening to mouse actions gets ticked, it tells the camera to change position.
6	//Tick Graphic s System	System (Ticked)	Graphic s 3D System	In this design the graphics resides in its own component and will be "ticked" to tell the graphics system to operate on a list of objects.
7	//Get Views and Visible Objects	Graphic s 3D System	Object & Object Manage ment System (Data)	Get the objects that are visible for drawing.
8	//Draw views using updated data (the camera was updated)	Graphic s 3D System	Graphic s 3D System	Draw the returned objects to the screen based on the view context.

A - 1.1.1.3.1.3.1.1.3 Gather Resources

Type: *public* **UseCase**

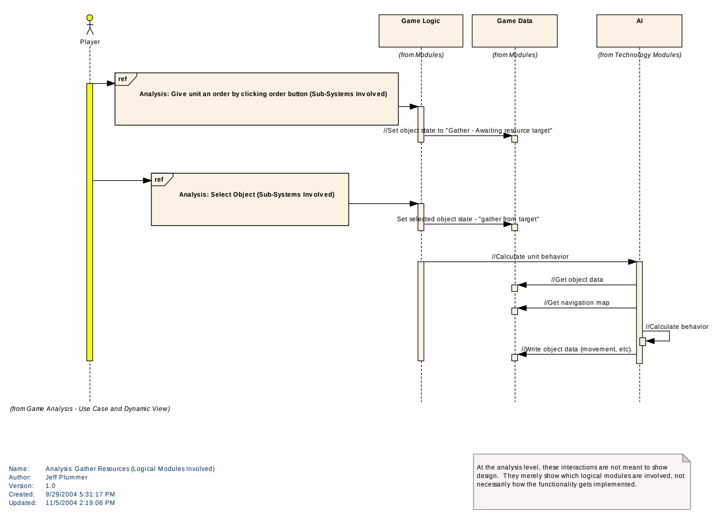
Package: Play Starcraft

Certain units can gather resources from the map. They interact with a resource object, and then carry some resources back to their base where it is added to the player's resources.

Scenarios

Basic Path {Basic Path}.

1. Unit receives gather resources command message (includes target).
2. Unit Moves to resource location.
3. Unit intereacts with resource for a period of time.
4. Unit moves to base.
5. Unit interacts with base, depositing the collected resources.



This diagram shows what logical modules are required to perform the "Gather Resources" use-case.

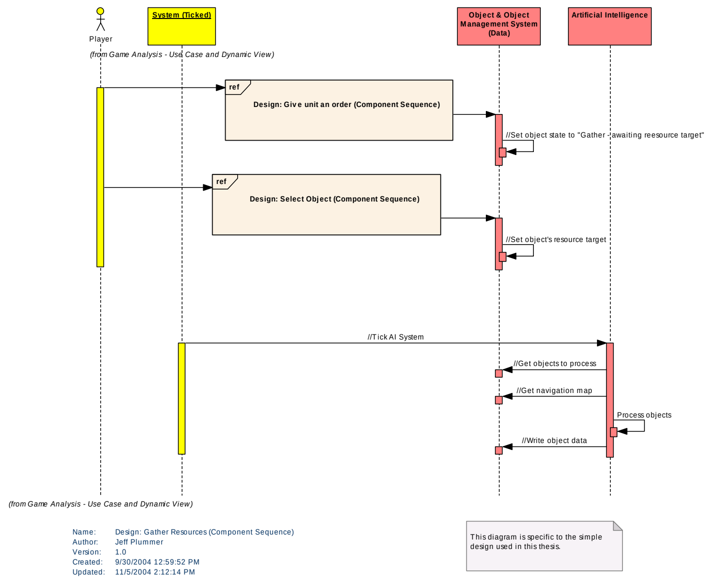
Figure 56 : Analysis: Gather Resources (Logical Modules Involved)

Analysis: Gather Resources (Logical Modules Involved) Messages

ID	Message	From Object	To Object	Notes
1		Player	Analysis: Give unit an order by	Call the "Analysis: Give unit an order" use case

			clicking order button (Sub-Systems Involved)	
2		Analysis: Give unit an order by clicking order button (Sub-Systems Involved)	Game Logic	Represents the beginning of the details specific to this "Give Unit an Order" interaction.
3	//Set object state to "Gather - Awaiting resource target"	Game Logic	Game Data	Game logic tells the object to prepare for an input selecting the resource target for that unit to gather from.
4		Player	Analysis: Select Object (Sub-Systems Involved)	Call the "Analysis: Select Object" use case
5		Analysis: Select Object (Sub-Systems Involved)	Game Logic	Represents the beginning of the details specific to this "Select Object" interaction.
6	Set selected object	Game Logic	Game Data	Tell the object waiting for a gather target, to target the resource that has just

	state - "gather from target"			been selected.
7	//Calculate unit behavior	Game Logic	AI	The AI logical module will determine how the object should behave - in this case how the object will behave in order to gather resources.
8	//Get object data	AI	Game Data	The AI functionality requires object data to process like current position, target resource position, etc.
9	//Get navigation map	AI	Game Data	The AI functionality requires map navigation data to process. The navigation map is data that says how an object can move from one location to another.
10	//Calculate behavior	AI	AI	Using the object data and map information the AI logical module will determine what the object should do. It will decide how the object should move to the resource(if required) etc.
11	//Write object data (movement, etc).	AI	Game Data	Once the AI functionality has decided what the object will do, the data / state information must be saved inside the object.



This diagram shows the sequence of events at the component level that occur to complete the "Gather Resources" use case.

Figure 57 : Design: Gather Resources (Component Sequence)

Design: Gather Resources (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1		Player	Design: Give unit an order (Component Sequence)	Call the "Give Unit an Order" use case.
2		Design: Give unit an order (Component Sequence)	Object & Object Management System (Data)	Represents the beginning of the order specific interaction details.
3	//Set	Object	Object	In this design the game

	object state to "Gather - awaiting reesource target"	& Object Management System (Data)	& Object Management System (Data)	logic resides within the game object itself, so the object readies itself for receiving the resource target.
4		Player	Design: Select Object (Component Sequence)	Call the "Select Object" use case.
5		Design: Select Object (Component Sequence)	Object & Object Management System (Data)	Represents the beginning of the details specific to this "Select Object" interaction.
6	//Set object's resource target	Object & Object Management System (Data)	Object & Object Management System (Data)	In this design the game logic resides within the game object itself, the object sets the resource target to the object that has just been selected.
7	//Tick AI System	System (Ticked)	Artificial Intelligence	In this design the AI resides in its own component and will be "ticked" to tell the AI system to operate on a list of objects.
8	//Get objects to process	Artificial Intelligence	Object & Object Management System (Data)	In this design the AI system will request list(s) of objects to process. The Object management component is responsible for providing the domain-specific component list(s) of relavant objects (i.e. not ALL the objects).

9	//Get navigation map	Artificial Intelligence	Object & Object Management System (Data)	The AI system will require some form of traversability map of the object system. The traversability map is important so the object can travel to the resource target.
10	Process objects	Artificial Intelligence	Artificial Intelligence	Using the object data and map information the AI component will determine what the object should do. It will decide how the object should move (if required) and any gather resource specific behavior.
11	//Write object data	Artificial Intelligence	Object & Object Management System (Data)	Update the object with data specific to the gather behavior the AI component decided.

A - 1.1.1.3.1.3.1.1.4 Give unit an order

Type: *public* **UseCase**
Package: Play Starcraft

Orders can vary from move, attack, patrol, etc.

Scenarios

Order With Target. {Basic Path}.
DESCRIPTION:
The most common orders are move, attack, and gather resources order. After giving the order, the target of the order must be selected. E.g. Giving a unit a move command requires the user to select the location to where the unit should move.

1. User clicks the order proper order command from the command window.
2. User selects the target of the order

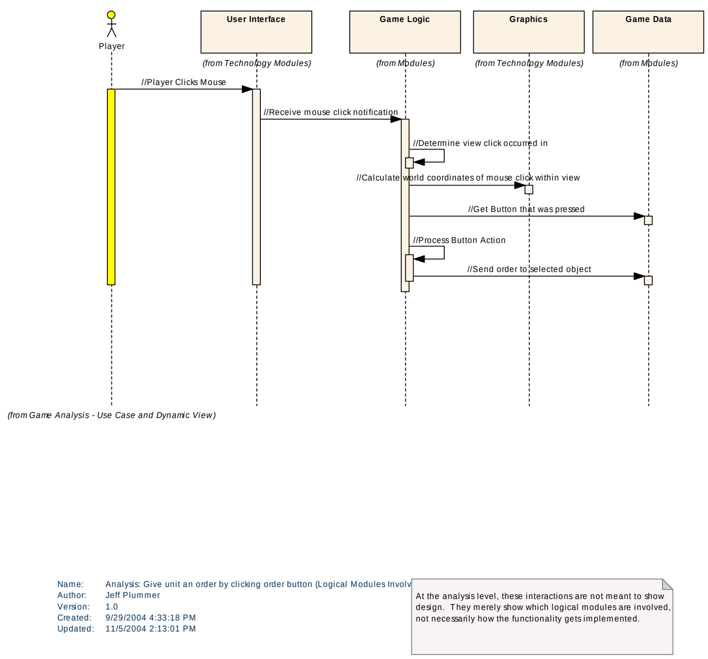
Fast order command {Alternate}.

DESCRIPTION:
Right clicking a location represents the fast order command.

1. User right clicks a location on the main map display.
2. Unit evaluates the order command. E.g. Right clicking an empty location will mean execute the MOVE command, right clicking an enemy unit will imply the ATTACK command.

Order with invalid target {Exceptional}.
DESCRIPTION:
User selects an invalid target of an order. E.g. User orders the unit to gather resources from a non-resource object, or empty location.

1. Notify user of invalid order.
2. Abort order - return to state before order was given.

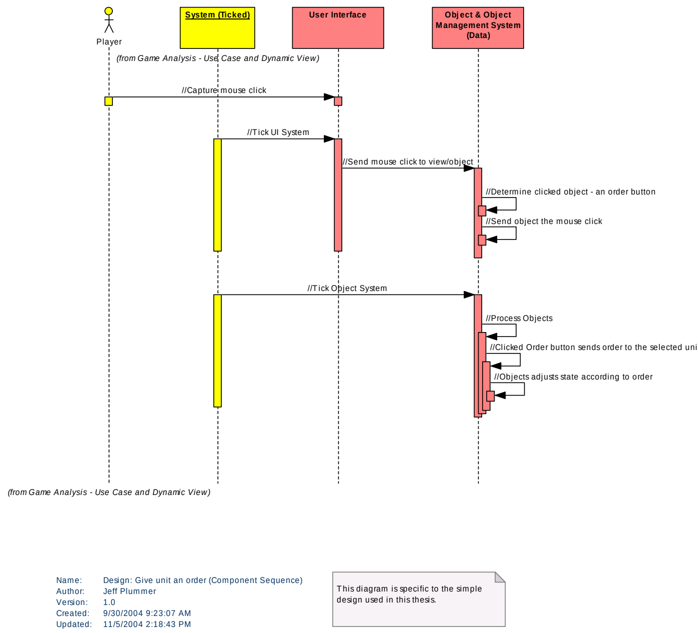


This diagram shows what logical modules are required to perform the "Give Unit an Order" use-case.

Figure 58 : Analysis: Give unit an order by clicking order button (Logical Modules Involved)

Analysis: Give unit an order by clicking order button (Logical Modules Involved) Messages

ID	Message	From Object	To Object	Notes
1	//Player Clicks Mouse	Player	User Interface	Player clicks mouse cursor over an object
2	//Receive mouse click notification	User Interface	Game Logic	Pass the mouse click to the game logic to determine the action .
3	//Determine view click occurred in	Game Logic	Game Logic	Determine which game view was clicked... in this case its the command button view.
4	//Calculate world coordinates of mouse click within view	Game Logic	Graphics	Get the screen coordinates of the objects
5	//Get Button that was pressed	Game Logic	Game Data	Get the button object that is at the screen location.
6	//Process Button Action	Game Logic	Game Logic	Determine order that was clicked
7	//Send order to selected object	Game Logic	Game Data	Send order command to the selected object.



This diagram shows the sequence of events at the component level that occur to complete the "Give unit an order" use case.

Figure 59 : Design: Give unit an order (Component Sequence)

Design: Give unit an order (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Capture mouse click	Player	User Interface	
2	//Tick UI System	System (Ticked)	User Interface	Ticking the UI system tells the UI Component to grab the status of all UI devices, or atleast update variables based on the UI events that occurred.
3	//Send mouse click to view/object	User Interface	Object & Object Management System	Tell any UI event listening objects in the game object component about the mouse movement.

			(Data)	
4	//Determine clicked object - an order button	Object & Object Management System (Data)	Object & Object Management System (Data)	In this simple design, when the graphics engine is ticked the graphics engine updates the screen coordinate position. We then use that value to determine which object was clicked.
5	//Send object the mouse click	Object & Object Management System (Data)	Object & Object Management System (Data)	Tell the object it's been clicked. It will process the click when the object itself is ticked.
6	//Tick Object System	System (Ticked)	Object & Object Management System (Data)	In this simple design the game logic resides in the object system, so ticking the object component is the same as processing all game logic.
7	//Process Objects	Object & Object Management System (Data)	Object & Object Management System (Data)	Objects are processesed in batch performing the game logic.
8	//Clicked Order button sends order to the selected unit	Object & Object Management System (Data)	Object & Object Management System (Data)	The game logic for the button object is to send an order command to the selected object.
9	//Objects adjusts state according to order	Object & Object Management System (Data)	Object & Object Management System (Data)	When the object receives an order command, it sets its state accordingly.

A - 1.1.1.3.1.3.1.1.5 Move to Location

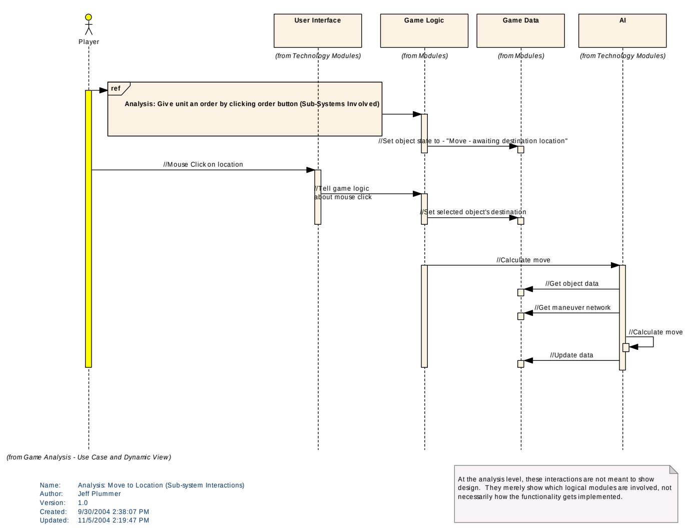
Type: *public* **UseCase**
Package: Play Starcraft

Tell an object to move from it's current location to a specific destination.

Scenarios

Order with target destination {Basic Path}.
After giving the order, the target destination of the order "move" must be selected.

1. Player clicks the move button.
2. Player clicks a destination location on the map.



This diagram shows what logical modules are required to perform the "Move to Location" use-case.

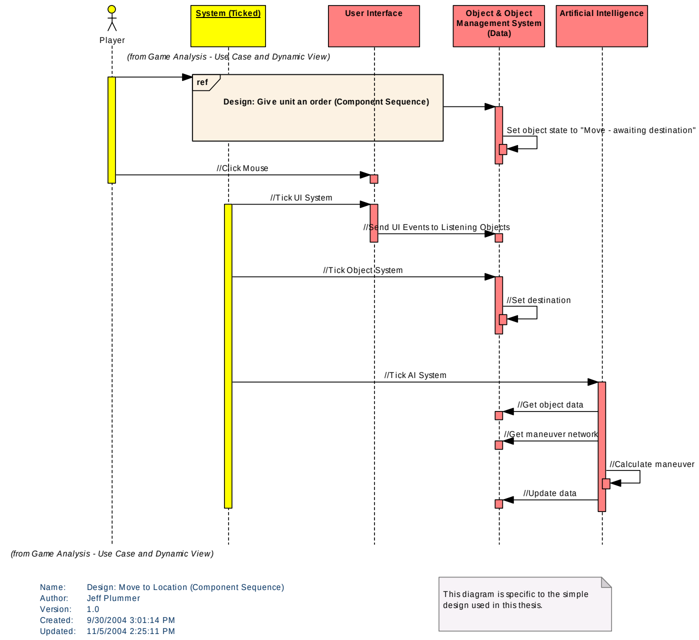
Figure 60 : Analysis: Move to Location (Sub-system Interactions)

Analysis: Move to Location (Sub-system Interactions) Messages

ID	Message	From Object	To Object	Notes
1		Player	Analysi	Call the "Analysis: Give

			s: Give unit an order by clicking order button (Sub-Systems Involved)	unit an order" use case
2		Analysis: Give unit an order by clicking order button (Sub-Systems Involved)	Game Logic	Represents the beginning of the details specific to this "Give Unit an Order" interaction.
3	//Set object state to - "Move - awaiting destination location"	Game Logic	Game Data	Game logic tells the object to prepare for an input selecting the target locatio for that unit to move to.
4	//Mouse Click on location	Player	User Interface	Player clicks on movement destination on the map.
5	//Tell game logic about mouse click	User Interface	Game Logic	The game logic will process the mouse click
6	//Set selected object's	Game Logic	Game Data	Tell the object waiting for a movement target, to target the location that

	destinat ion			has just been clicked.
7	//Calcul ate move	Game Logic	AI	The AI logical module will determine how the object should behave - in this case how the object will behave in order to move from one location to another.
8	//Get object data	AI	Game Data	The AI functionality requires object data to process like current position, target position, etc.
9	//Get maneuv er network	AI	Game Data	The AI functionality requires map navigation data to process. The navigation map is data that says how an object can move from one location to another.
1 0	//Calcul ate move	AI	AI	Using the object data and map information the AI logical module will determine what the object should do. It will decide the path the object will travel to the destination.
1 1	//Updat e data	AI	Game Data	Once the AI functionality has decided what the object will do, the data / state information must be saved inside the object.



This diagram shows the sequence of events at the component level that occur to complete the "Move to Location" use case.

Figure 61 : Design: Move to Location (Component Sequence)

Design: Move to Location (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1		Player	Design: Give unit an order (Component Sequence)	Call the "Give Unit an Order" use case.
2		Design: Give unit an order (Component Sequence)	Object & Object Management System (Data)	Represents the beginning of the order specific interaction details.

3	Set object state to "Move - awaiting destination"	Object & Object Management System (Data)	Object & Object Management System (Data)	In this design the game logic resides within the game object itself, so the object readies itself for receiving the movement destination.
4	//Click Mouse	Player	User Interface	The player clicks the mouse button on a location on the map.
5	//Tick UI System	System (Ticked)	User Interface	Ticking the UI system tells the UI Component to grab the status of all UI devices, or atleast update variables based on the UI events that occurred.
6	//Send UI Events to Listening Objects	User Interface	Object & Object Management System (Data)	Tell any UI event listening objects in the game object component about the mouse movement.
7	//Tick Object System	System (Ticked)	Object & Object Management System (Data)	In this simple design the game logic resides in the object system, so ticking the object component is the same as processing all game logic.
8	//Set destination	Object & Object Management System (Data)	Object & Object Management System (Data)	The game logic in the listener object sets the selected object's movement destination.
9	//Tick AI System	System (Ticked)	Artificial Intelligence	In this design the AI resides in its own component and will be "ticked" to tell the AI system to operate on a list of objects.
1	//Get	Artificia	Object	In this design the AI

0	object data	l Intelligence	& Object Management System (Data)	system will request list(s) of objects to process. The Object management component is responsible for providing the domain-specific component list(s) of relavant objects (i.e. not ALL the objects).
11	//Get maneuver network	Artificial Intelligence	Object & Object Management System (Data)	The AI system will require some form of traversability map of the object system. The traversability map is important so the object can travel to the resource target.
12	//Calculate maneuver	Artificial Intelligence	Artificial Intelligence	Using the object data and map information the AI component will determine what the object should do. It will decide how the object should move.
13	//Update data	Artificial Intelligence	Object & Object Management System (Data)	Update the object with data specific to the gather behavior the AI component decided.

A - 1.1.1.3.1.3.1.1.6 Research Technology

Type: public **UseCase**
Package: Play Starcraft

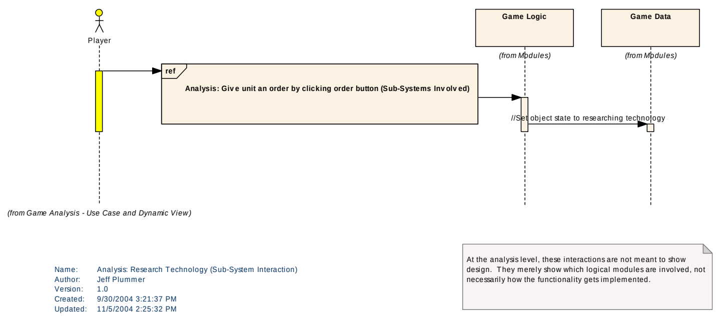
New technologies that enhance units, or provide new unit actions can be researched by many buildings. Research takes time and resources.

Scenarios

Click research command button {Basic Path}.

1. Player clicks the research command.
2. Unit begins a timed research process.

3. Research timer is updated during research process.
4. Upon completion, the unit is upgraded with the new ability.



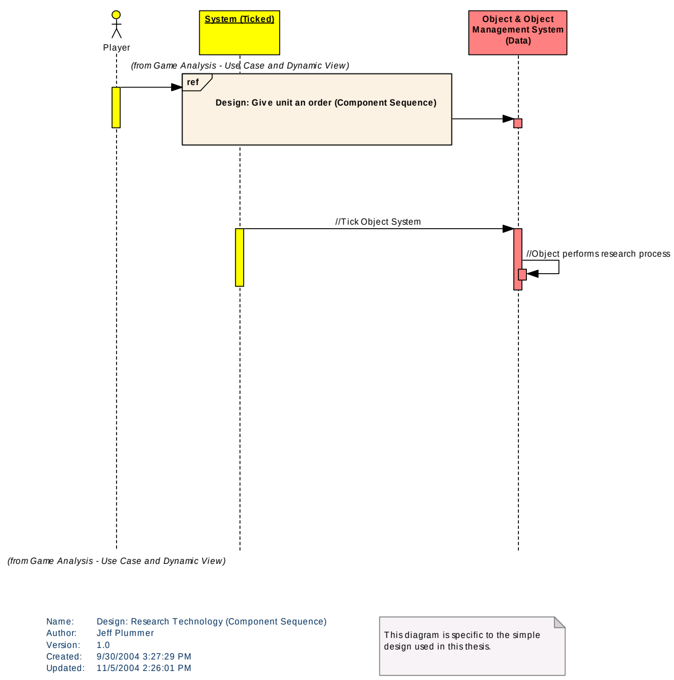
This diagram shows what logical modules are required to perform the "Research Technology" use-case.

Figure 62 : Analysis: Research Technology (Sub-System Interaction)

Analysis: Research Technology (Sub-System Interaction) Messages

ID	Message	From Object	To Object	Notes
1		Player	Analysis: Give unit an order by clicking order button (Sub-Systems Involved)	Call the "Analysis: Give unit an order" use case
2		Analysis: Give unit an order by clicking order button (Sub-Systems	Game Logic	Represents the beginning of the details specific to this "Give Unit an Order" interaction.

		Involved)		
3	//Set object state to researching technology	Game Logic	Game Data	Game logic tells the object to begin researching a technology



This diagram shows the sequence of events at the component level that occur to complete the "Research technology" use case.

Figure 63 : Design: Research Technology (Component Sequence)

Design: Research Technology (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1		Player	Design: Give	Call the "Give Unit an Order" use case.

			unit an order (Component Sequence)	
2		Design: Give unit an order (Component Sequence)	Object & Object Management System (Data)	Represents the beginning of the order specific interaction details.
3	//Tick Object System	System (Ticked)	Object & Object Management System (Data)	In this simple design the game logic resides in the object system, so ticking the object component is the same as processing all game logic.
4	//Object performs research process	Object & Object Management System (Data)	Object & Object Management System (Data)	Ticking the object that is performing technology research sets it to increase its research progress.

A - 1.1.1.3.1.3.1.1.7 Select Object

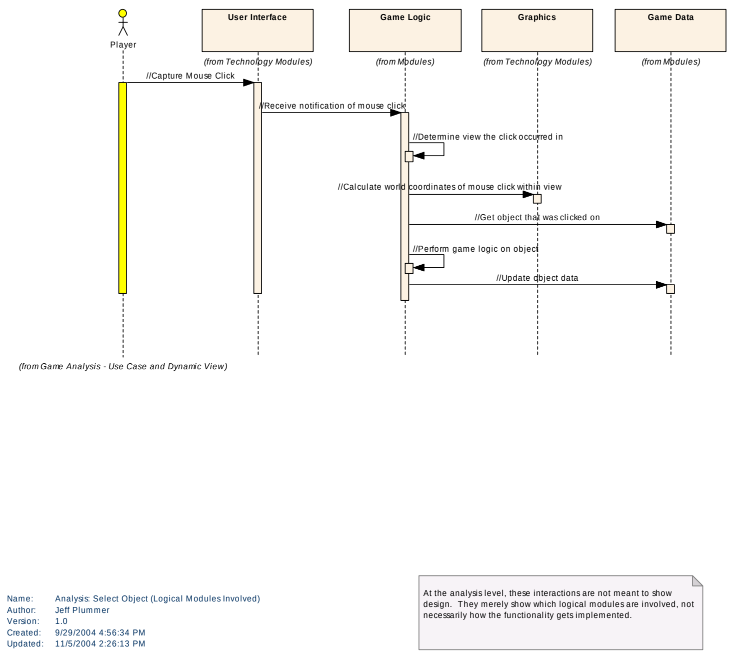
Type: *public* **UseCase**
Package: Play Starcraft

Selectable objects include units, buildings, resources, and wild creatures.

Scenarios

Mouse click on unit {Basic Path}.
Description: Player clicks the left mouse button over a unit in the main view screen.

1. Tell unit is has been selected.
2. Tell views that an object(s) has been selected.



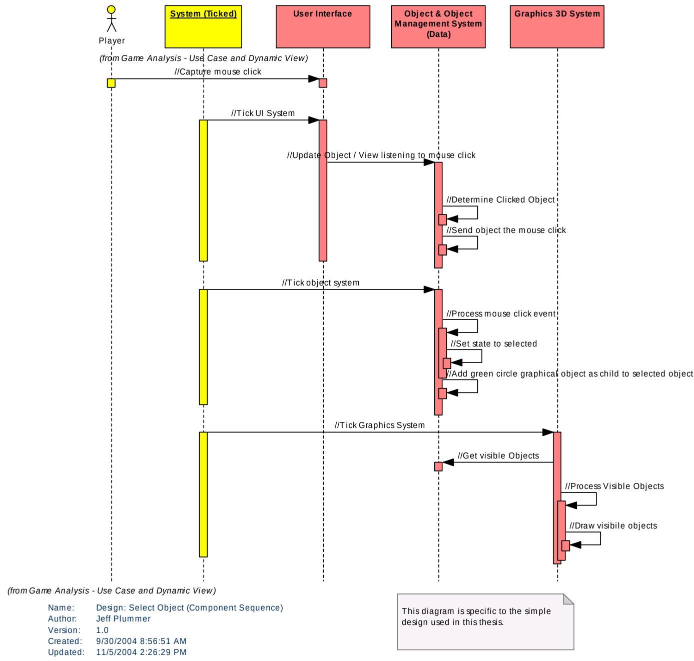
This diagram shows what logical modules are required to perform the "Select Object" use-case.

Figure 64 : Analysis: Select Object (Logical Modules Involved)

Analysis: Select Object (Logical Modules Involved) Messages

ID	Message	From Object	To Object	Notes
1	//Capture Mouse Click	Player	User Interface	Players clicks the mouse over the graphical representation of a game object.
2	//Receive notification of mouse click	User Interface	Game Logic	Pass the mouse click to the game logic to determine the action .
3	//Determine	Game Logic	Game Logic	Determine which game view was clicked... in this

	view the click occurred in			case its the main game view.
4	//Calculate world coordinates of mouse click within view	Game Logic	Graphics	Get the screen coordinates of the objects
5	//Get object that was clicked on	Game Logic	Game Data	Get the game object that is at the screen location.
6	//Perform game logic on object	Game Logic	Game Logic	Tell the object it's been selected
7	//Update object data	Game Logic	Game Data	Update the data so it is drawn with a green circle around it, and mark it as reciever of any new orders.



This diagram shows the sequence of events at the component level that occur to complete the "Select Object" use case.

Figure 65 : Design: Select Object (Component Sequence)

Design: Select Object (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Capture mouse click	Player	User Interface	
2	//Tick UI System	System (Ticked)	User Interface	Ticking the UI system tells the UI Component to grab the status of all UI devices, or atleast update variables based on the UI events that occurred.
3	//Update Object / View listening to	User Interface	Object & Object Management System	Tell any UI event listening objects in the game object component about the mouse movement.

	mouse click		(Data)	
4	//Determine Clicked Object	Object & Object Management System (Data)	Object & Object Management System (Data)	In this simple design, when the graphics engine is ticked the graphics engine updates the screen coordinate position. We then use that value to determine which object was clicked.
5	//Send object the mouse click	Object & Object Management System (Data)	Object & Object Management System (Data)	Tell the object it's been clicked. It will process the click when the object itself is ticked.
6	//Tick object system	System (Ticked)	Object & Object Management System (Data)	In this simple design the game logic resides in the object system, so ticking the object component is the same as processing all game logic.
7	//Process mouse click event	Object & Object Management System (Data)	Object & Object Management System (Data)	Objects are processesed in batch performing the game logic.
8	//Set state to selected	Object & Object Management System (Data)	Object & Object Management System (Data)	The object that received the mouse click processes it to set it's state to selected.
9	//Add green circle graphical object as child	Object & Object Management System (Data)	Object & Object Management System (Data)	Set drawing info to say it has a green circle around it.

	to selected object			
10	//Tick Graphics System	System (Ticked)	Graphics 3D System	Ticking the graphics system tells the graphics Component to draw all visible objects on the screen.
11	//Get visible Objects	Graphics 3D System	Object & Object Management System (Data)	Get the views and object lists to draw.
12	//Process Visible Objects	Graphics 3D System	Graphics 3D System	process the visible object as a batch.
13	//Draw visibile objects	Graphics 3D System	Graphics 3D System	Draw the objects based on their graphics data, and the view context.

A - 1.1.1.3.1.3.1.1.8 Building construct Unit

Type: *public* **UseCase**
Package: Play Starcraft

This unique order cause a building to construct a unit

A - 1.1.1.3.1.3.1.1.9 Give Building an order

Type: *public* **UseCase**
Package: Play Starcraft

Most buildings have the ability to carry out certain orders like constructing military units, or performing research.

A - 1.1.1.3.1.3.1.1.10 Hold Position

Type: *public* **UseCase**
Package: Play Starcraft

Orders unit to stay at its current location. Do not follow enemies to attack them.

A - 1.1.1.3.1.3.1.1.11 Manipulate Object Resources

Type: *public* **UseCase**
Package: Play Starcraft

When you mine a resource (from a gyser object or a crystal object), you are reducing the amount of resources available in that object.

A - 1.1.1.3.1.3.1.1.12 Manipulate Player Resources

Type: *public* **UseCase**
Package: Play Starcraft

Add / subtract from the resources the player has. Resources are used as "money" to build and research.

A - 1.1.1.3.1.3.1.1.13 Modify Doable Commands

Type: *public* **UseCase**
Package: Play Starcraft

Display icons representing the commands available to the user at this time.

A - 1.1.1.3.1.3.1.1.14 Patrol Location

Type: *public* **UseCase**
Package: Play Starcraft

Unit will move back and forth in an attack ready state between the objects current location and the target destination.

A - 1.1.1.3.1.3.1.1.15 Stop Movement

Type: *public* **UseCase**
Package: Play Starcraft

Orders a unit to halt its current movement command.

A - 1.1.1.3.1.3.1.1.16 Unit Construct Building

Type: *public* **UseCase**

Package: Play Starcraft

This unique order causes a unit to construct a building

A - 1.1.1.3.1.4 Design: Tick Starcraft System

The artifacts contained within this package show many of the architectural independent artifcats reworked using the simple proposed design. They are merely meant to show another view into how logic flows using the proposed architecture, and the simple design.

This diagram shows the various "ticking" of the different domain-specific systems to create the game of Starcraft(tm).

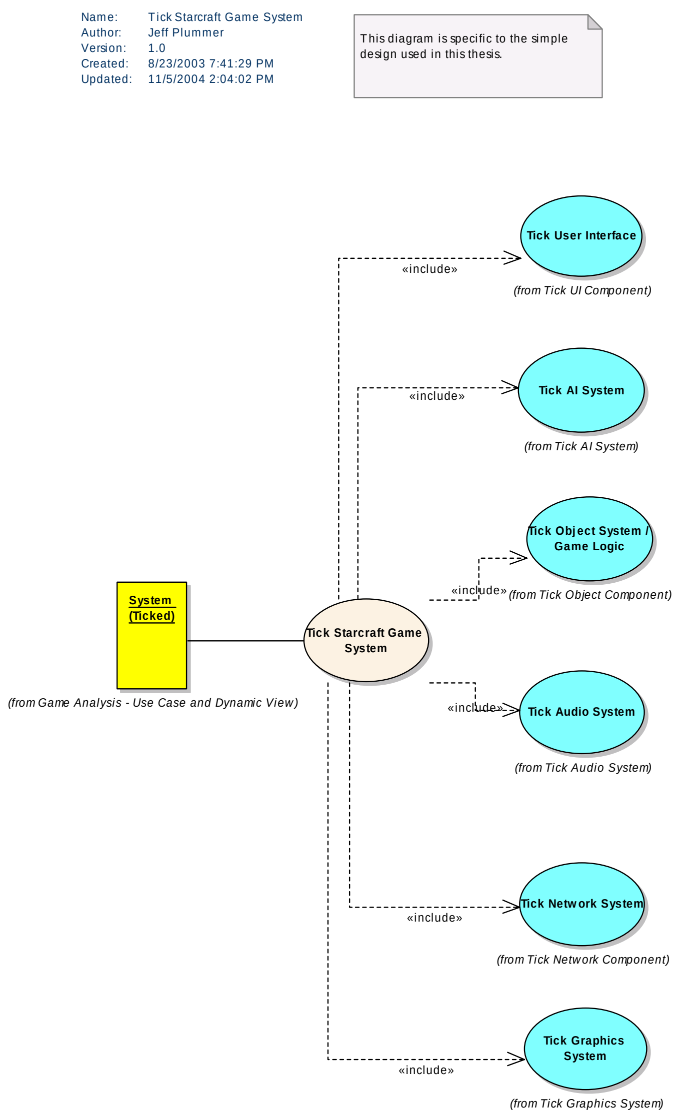


Figure 66 : Tick Starcraft Game System

A - 1.1.1.3.1.4.1.1.1 Tick Starcraft Game System

Type: *public* **UseCase**
Package: Design: Tick Starcraft System

This design dependent use case represents the process of ticking all the domain-specific components to create the game behavior.

A - 1.1.1.3.1.4.2   Tick AI System

The diagram shows the use cases involved in the ticking of the AI component that is needed for the game of Starcraft(tm).

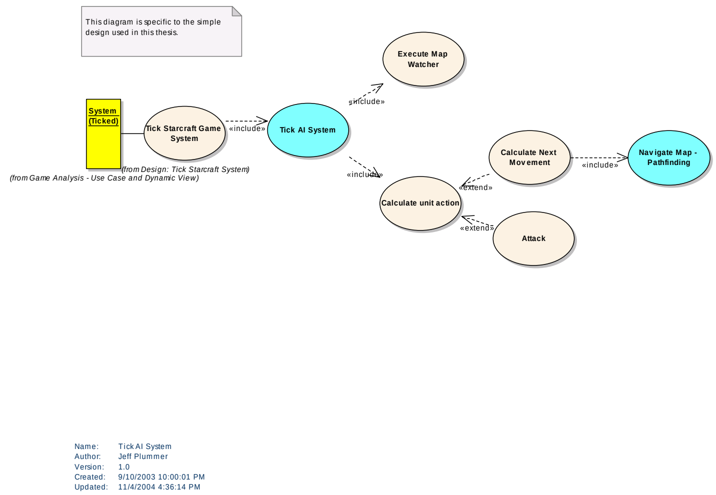


Figure 67 : Tick AI System

A - 1.1.1.3.1.4.2.1.1 Tick AI System

Type: *public* **UseCase**
Package: Tick AI System

Tick the artificial intelligence component. Execute AI operations that determine what the the objects intend to do next.

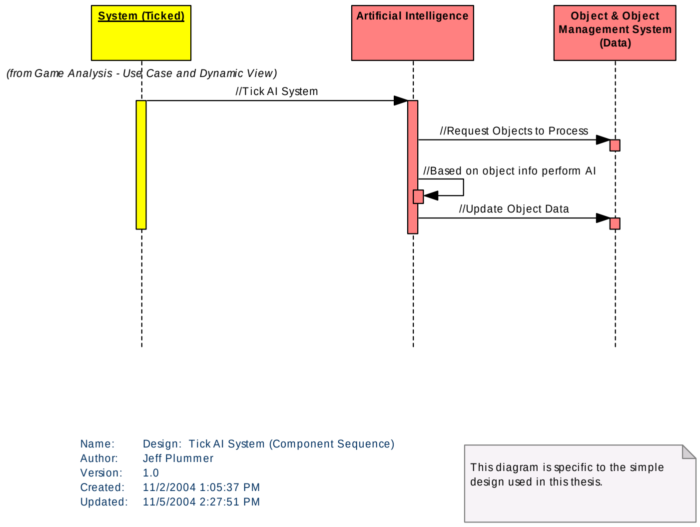
Starcraft AI system will determine computer players' decisions, an object's next move, and some AI state information.

Scenarios

Tick AI System {Basic Path}.

1. Request views/object lists of AI objects to process.
2. Read object AI related information (state, etc.).
3. Process objects.

NOTE: Objects don't exist in a vacuum. The AI system could provide messaging, etc. for AI interactions to take place between objects.



This diagram shows the sequence of events at the component level that occur to complete the "Tick AI System" use case.

Figure 68 : Design: Tick AI System (Component Sequence)

Design: Tick AI System (Component Sequence) Messages

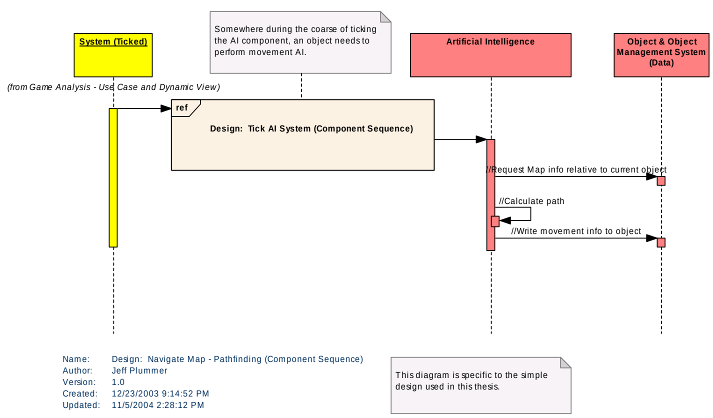
ID	Message	From Object	To Object	Notes
1	//Tick AI System	System (Ticked)	Artificial Intelligence	In this design the AI resides in its own component and will be "ticked" to tell the AI system to operate on a list of objects.
2	//Request Objects to Process	Artificial Intelligence	Object & Object Management System (Data)	Request list(s) of objects that require AI processing.
3	//Based on object info perform	Artificial Intelligence	Artificial Intelligence	Objects are like state machines and depending on their state, different types AI processing will be done for each object.

	AI			
4	//Update Object Data	Artificial Intelligence	Object & Object Management System (Data)	After AI object processing has completed, update the object data.

A - 1.1.1.3.1.4.2.1.2 Navigate Map - Pathfinding

Type: *public* **UseCase**
Package: Tick AI System

This represents the process that will analyze the map, and provide a potential path to a location.



This diagram shows the sequence of events at the component level that occur to complete the "Navigate Map" use case.

Figure 69 : Design: Navigate Map - Pathfinding (Component Sequence)

Design: Navigate Map - Pathfinding (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1		System (Ticked	Design: Tick AI	

		)	System (Component Sequence)	
2		Design: Tick AI System (Component Sequence)	Artificial Intelligence	During the coarse of ticking the AI system, an object requires AI to move, and therefore needs to perform pathfinding AI.
3	//Request Map info relative to current object	Artificial Intelligence	Object & Object Management System (Data)	The object system has map and traversability information.
4	//Calculate path	Artificial Intelligence	Artificial Intelligence	Perform pathfinding AI to determine the movement path the object should take.
5	//Write movement info to object	Artificial Intelligence	Object & Object Management System (Data)	Update the object with the movement information.

A - 1.1.1.3.1.4.2.1.3 Attack

Type: *public* **UseCase**
Package: Tick AI System

When an object performs an "attack" action, the object will cycle and send out an attack message, that should remove hitpoints, etc.

A - 1.1.1.3.1.4.2.1.4 Calculate AI State

Type: *public* **UseCase**
Package: Tick AI System

A - 1.1.1.3.1.4.2.1.5 Calculate Next Movement

Type: *public* **UseCase**
Package: Tick AI System

Determine the object's next movement direction. This depends on the object's state and destination. For example if the unit is resource gathering it's movement should sort of "wander" around the resource gathering resources. If it's a move or attack command state, it should move in the fastest path to the target.

Example states:

1. MOVE (move toward the target location)
2. MOVE_TO_ATTACK (move toward the target object)
3. ATTACKING (Object shouldn't move)

An example of an object chasing another object attacking it, the object would change states from MOVE_TO_ATTACK to ATTACKING back and forth while it attacks the fleeing creature.

A - 1.1.1.3.1.4.2.1.6 Calculate unit action

Type: *public* **UseCase**
Package: Tick AI System

For each unit, review the object state and determine it's next course of action.

The map watcher may have put the object in an "attack" state or "move to attack" state in which it will move or attack.

A - 1.1.1.3.1.4.2.1.7 Execute Map Watcher

Type: *public* **UseCase**
Package: Tick AI System

NOTE: This is just one possible way of doing things.

Map watcher tracks object zones. Objects register zones to watch, when an object enters their zone, they receive a message.

All objects will register a zone of sight so they receive messages when an enemy becomes visible.

Objects will also register "attack" and "move to attack" zones. When an enemy enters the "move to attack" zone, the object will move toward the object until it enters it's "attack" zone. When an object enters it's "attack" zone it attacks.

A - 1.1.1.3.1.4.3 Tick Audio System

The diagram shows the use cases involved in the ticking of the Audio component that is needed for the game of Starcraft(tm).

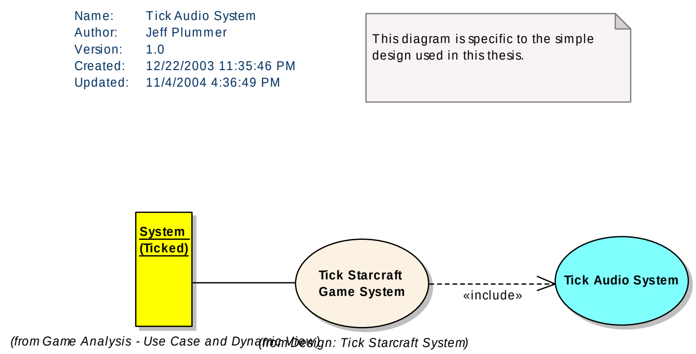


Figure 70 : Tick Audio System

A - 1.1.1.3.1.4.3.1.1 Tick Audio System

Type: *public* **UseCase**
Package: Tick Audio System

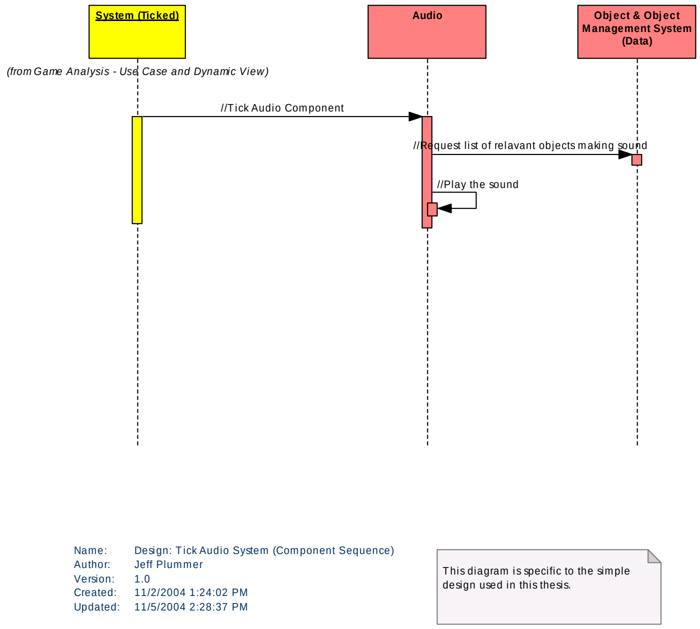
Tick the audio component. Play Background music, and play sound effects that have been signaled.

1. Get List of objects making sounds from obj component.
2.

Scenarios

Tick Audio System {Basic Path}.

1. Request views/object lists of Audio objects to process.
2. Read object Audio related information (state, etc.).
3. Enque sounds



This diagram shows the sequence of events at the component level that occur to complete the "Tick Audio System" use case.

Figure 71 : Design: Tick Audio System (Component Sequence)

Design: Tick Audio System (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick Audio Component	System (Ticked)	Audio	In this design the Audio resides in its own component and will be "ticked" to tell the Audio system to operate on a list of objects.
2	//Request list of relavant objects making sound	Audio	Object & Object Management System (Data)	Request list of objects near the "camera" that are currently making sounds.
3	//Play the sound	Audio	Audio	Send the sounds to the sound card

A - 1.1.1.3.1.4.4 Tick Graphics System

The diagram shows the use cases involved in the ticking of the Graphics component that is needed for the game of Starcraft(tm).

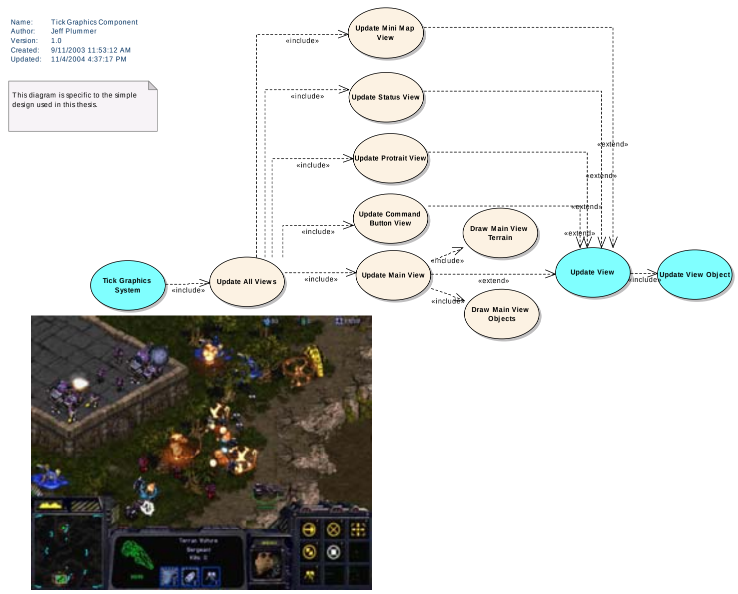


Figure 72 : Tick Graphics Component

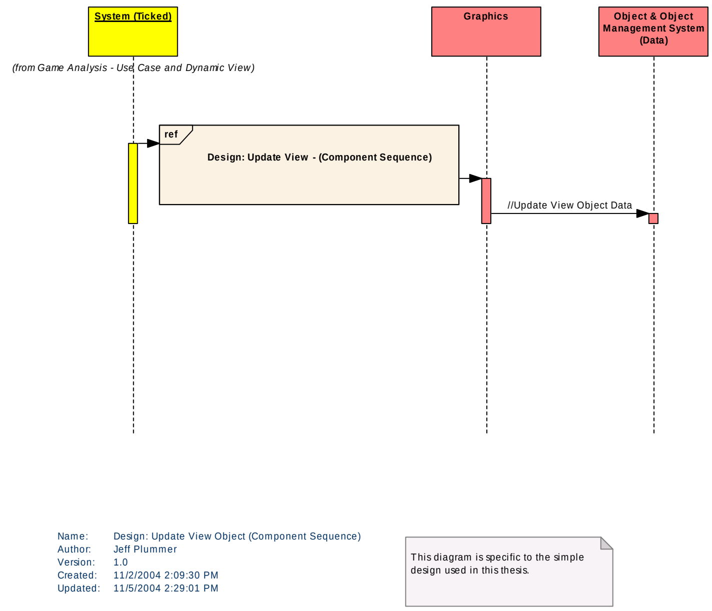
A - 1.1.1.3.1.4.4.1.1 <u>:IGraphicsObjectSystem</u>

Type: *public «interface»* **<u>Sequence instance : (IGraphicsObjectSystem)</u>**
Package: Tick Graphics System

A - 1.1.1.3.1.4.4.1.2 <u>Update View Object</u>

Type: *public* **<u>UseCase</u>**
Package: Tick Graphics System

This use case represents the functionality required to update each individual object visible in the view.



This diagram shows the sequence of events at the component level that occur to complete the "Update View Object" use case.

Figure 73 : Design: Update View Object (Component Sequence)

Design: Update View Object (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1		System (Ticked)	Design: Update View - (Component Sequence)	
2		Design: Update View - (Component Sequence)	Graphics	
3	//Update View	Graphics	Object &	Update things like animation state, screen

	Object Data		Object Management System (Data)	coordinates etc.

A - 1.1.1.3.1.4.4.1.3 Tick Graphics System

Type: *public* **UseCase**
Package: Tick Graphics System

Tick the graphics component. Draw whatever needs to be drawn.

Starcraft has several different viewports that need to be drawn, as well as the gui. Things like the main view, the minimap etc.

Scenarios

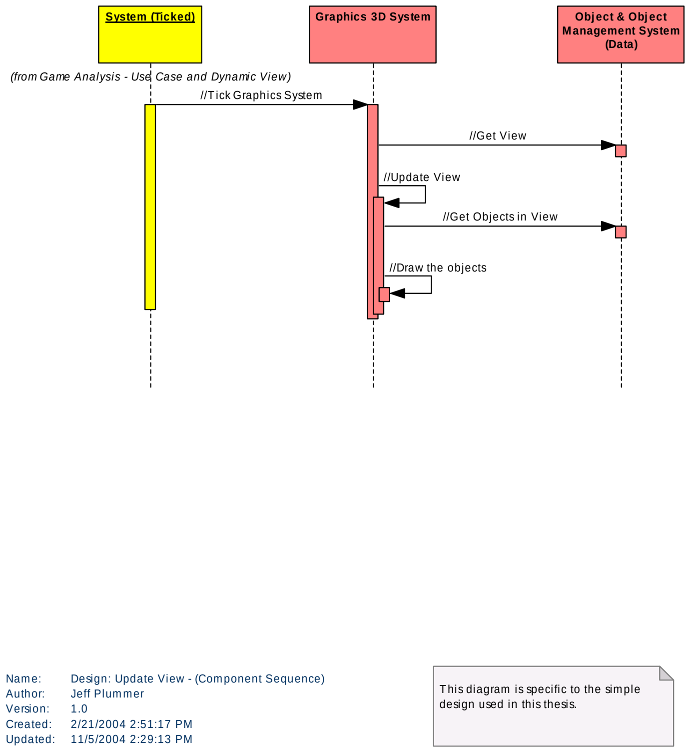
Tick Graphics System {Basic Path}.

1. Request views/object lists of objects to graphically process.
2. Read object graphics related information (position, graphics resource.).
3. Process/Draw objects.

A - 1.1.1.3.1.4.4.1.4 Update View

Type: *public* **UseCase**
Package: Tick Graphics System

Update view is the generic functionality that is extended by the specialized view updates.



This diagram shows the sequence of events at the component level that occur to complete the "Update View" use case.

Figure 74 : Design: Update View - (Component Sequence)

Design: Update View - (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick Graphics System	System (Ticked)	Graphics 3D System	In this design the graphics resides in its own component and will be "ticked" to tell the graphics system to operate on a list of objects.
2	//Get View	Graphics 3D System	Object & Object Management	Request view to be drawn.

			System (Data)	
3	//Update View	Graphics 3D System	Graphics 3D System	Update a view that is to be drawn
4	//Get Objects in View	Graphics 3D System	Object & Object Management System (Data)	Get the objects that are visible in that view.
5	//Draw the objects	Graphics 3D System	Graphics 3D System	Draw the objects using the view context.

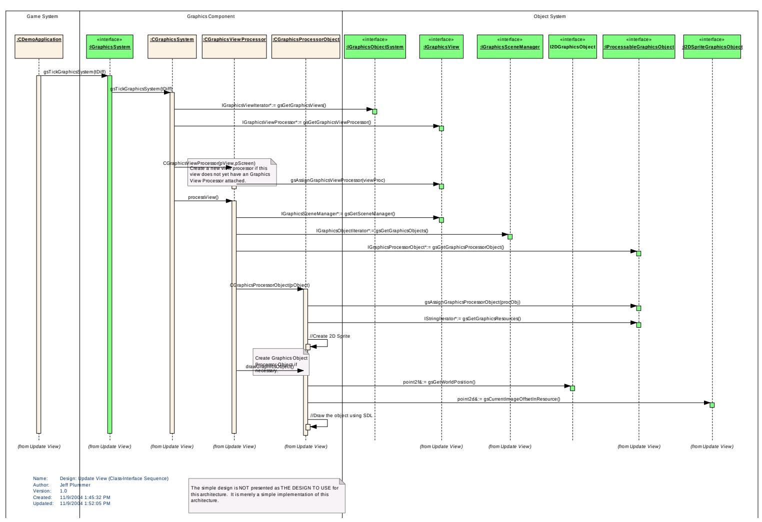


Figure 75 : Design: Update View (Class-Interface Sequence)

Design: Update View (Class-Interface Sequence) Messages

ID	Message	From Object	To Object	Notes
1	gsTick			Interface - Tick the

	GraphicsSystem(float)			Graphics system.
2	gsTickGraphicsSystem(float)			Implementation - Tick the Graphics System.
3	gsGetGraphicsViews()			Interface - Get Views of Graphics objects to process... This prototype only contains one view.
4	gsGetGraphicsViewProcessor()			Interface - Get the Graphics View Processor if it exists.
5	CGraphicsViewProcessor(IGraphicsView*, SDL_Surface*)			Create a view processor if this view does not yet have one - i.e. this is our first time processing this view.
6	gsAssignGraphicsViewProcessor(IGraphicsViewProcessor*)			Interface - Assign the view processor to the view.
7	processView()			Graphics Process the view
8	gsGetSceneManager()			Interface - Get the Scenemanager (structured list of objects to process)
9	gsGetGraphicsObjects()			Interface - Get Ordered list of objects to process.
10	gsGetGraphicsProcess			Interface - Get the Graphics object processor responsible for

	orObject()			processing this object.
11	CGraphicsProcessorObject(IProcessableGraphicsObject*)			Create Graphics Object Processor Object if necessary.
12	gsAssignGraphicsProcessorObject(IGraphicsProcessorObject*)			Interface - Assign the processor object to the game object.
13	gsGetGraphicsResources()			Interface - Get the Graphics Resource information required to draw the object in 2D.
14	//Create 2D Sprite			Create the entity using SDL to manage sprites.
15	drawGraphicsObject()			Perform Graphics Processing on this object
16	gsGetWorldPosition()		I2DGraphicsObject	Get the position of the 2D object
17	gsCurrentImageOffsetInResource()			Interface - Get the sprite offset in the 2D image
18	//Draw the object using SDL			Use SDL to blit the sprite

A - 1.1.1.3.1.4.4.1.5 Update Main View

Type: *public* **UseCase**
Package: Tick Graphics System

Represents the process of the main display window updating to display the current state of the game.

Scenarios

Basic {Basic Path}.

1. Get view frame (area to display).
2. Draw Terrain
3. Draw Objects

A - 1.1.1.3.1.4.4.1.6 Draw Main View Objects

Type: *public* **UseCase**
Package: Tick Graphics System

Draw the game objects over the background.

A - 1.1.1.3.1.4.4.1.7 Draw Main View Terrain

Type: *public* **UseCase**
Package: Tick Graphics System

Paint the terrain background on the screen.

A - 1.1.1.3.1.4.4.1.8 Update All Views

Type: *public* **UseCase**
Package: Tick Graphics System

The Starcraft game has many "views" displayed on the screen during game play. There is a mini-map view, the main game view, etc. Each of these views needs to be drawn.

A - 1.1.1.3.1.4.4.1.9 Update Command Button View

Type: *public* **UseCase**
Package: Tick Graphics System

This view contains buttons that represent all the commands available for the selected object(s).

A - 1.1.1.3.1.4.4.1.10 Update Mini Map View

Type: *public* **UseCase**
Package: Tick Graphics System

Update the small view that shows a miniature view of the entire game map.

A - 1.1.1.3.1.4.4.1.11 Update Protrait View

Type: *public* **UseCase**
Package: Tick Graphics System

Update the protrait view that shows a picture or animation of the currently selected object(s).

A - 1.1.1.3.1.4.4.1.12 Update Status View

Type: *public* **UseCase**
Package: Tick Graphics System

The status view shows the health and other stats of the currently selected object(s).

A - 1.1.1.3.1.4.5   Tick Network Component

The diagram shows the use cases involved in the ticking of the Network component that is needed for the game of Starcraft(tm).

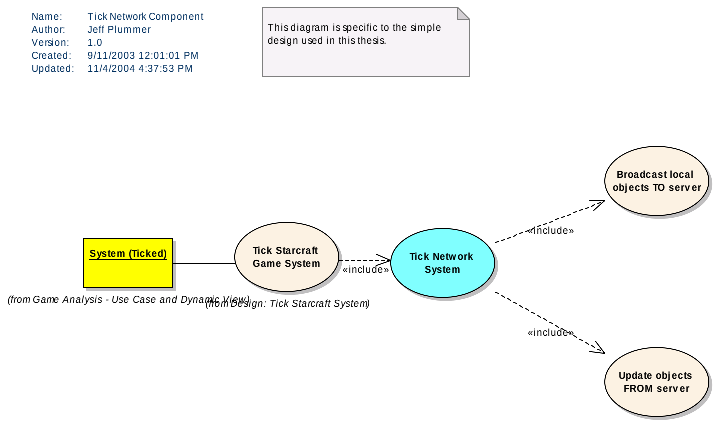


Figure 76 : Tick Network Component

A - 1.1.1.3.1.4.5.1.1 Broadcast local objects TO server

Type: *public* **UseCase**
Package: Tick Network Component

The actions of the player controlled objects are broadcast to the server, so other networked players can update their client machines.

Scenarios

Basic Path {Basic Path}.

1. Request necessary objects from Object Component -
   (This will become more clear in the component interfaces, but basically the object component will send a list of objects that are likely to be needed by the network).
2. Send relevant changes to server.

A - 1.1.1.3.1.4.5.1.2 Tick Network System

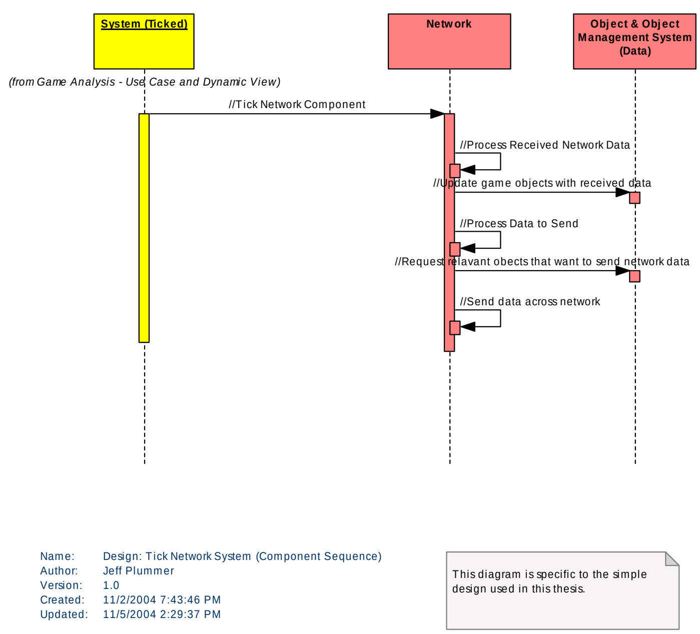
Type: *public* **UseCase**
Package: Tick Network Component

Send outgoing messages and respond to messages that have arrived from the network.

Scenarios

Tick Network System {Basic Path}.

1. Read received data.
2. Update objects with received data.
3. Request views/object lists of objects to write out to network.
4. Read object information to send (state, etc.).
5. Send info.



This diagram shows the sequence of events at the component level that occur to complete the "Tick Network System" use case.

Figure 77 : Design: Tick Network System (Component Sequence)

Design: Tick Network System (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick Network	System (Ticked)	Network	In this design the network resides in its own component and will be

	Compo nent			"ticked" to tell the network system to operate on a list of objects.
2	//Proces s Receive d Networ k Data	Networ k	Networ k	The network receiving of network data is in it's own thread, but act of doing something meaningful with the network data is performed in the main tick.
3	//Updat e game objects with receive d data	Networ k	Object & Object Manage ment System (Data)	Update the proper local objects with the updates that came from the network.
4	//Proces s Data to Send	Networ k	Networ k	
5	//Reque st relavant obects that want to send network data	Networ k	Object & Object Manage ment System (Data)	The object managment system will determine which objects are relavant to networked computers and should send their network data.
6	//Send data across network	Networ k	Networ k	Send the proper data across the network.

A - 1.1.1.3.1.4.5.1.3 Update objects FROM server

Type: *public* **UseCase**
Package: Tick Network Component

Data will arrive from the server detailing the actions of networked players actions. The network component will send the updates to the object component.

A - 1.1.1.3.1.4.6 Tick Object Component

The diagram shows the use cases involved in the ticking of the Object component (game logic) that is needed for the game of Starcraft(tm).

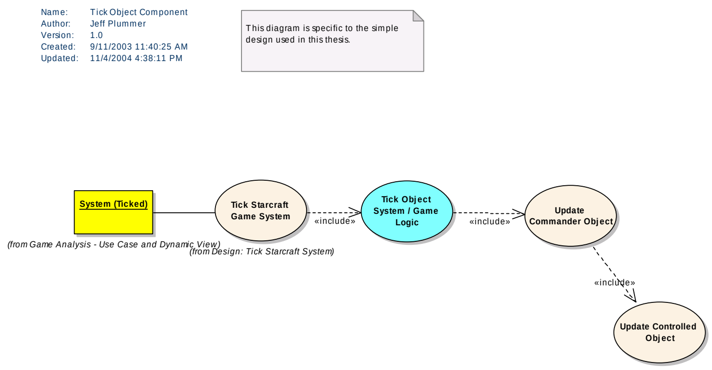


Figure 78 : Tick Object Component

A - 1.1.1.3.1.4.6.1.1 <u>Tick Object System / Game Logic</u>

Type: *public* **<u>UseCase</u>**
Package: Tick Object Component

Tick the object component. The object component is responsible for performing an actual action based on state information.

Starcraft's object system might evaluate the "Attacking" state and fire a bullet, change animation states, etc.

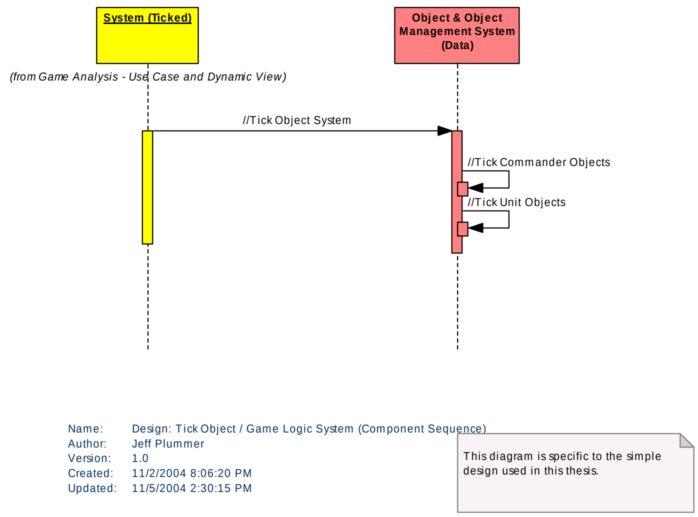
In this simple design game logic and object management exist in the same component. The object management portion updates the view structures so it will provide relavant object lists. The game logic portion performs some minor game logic processing of objects.

In hindsight this is bad, and game logic truly should be its own component. But since its a design, not architecture problem, it wasn't worth fixing in the prototype.

Scenarios

<u>Tick Object System / Game Logic</u> {Basic Path}.
1. Update Views / object lists
2. Request views/object lists of objects to process.
3. Read object game logic related information (state, etc.).

4. Process objects.



This diagram shows the sequence of events at the component level that occur to complete the "Tick Object / Game Logic System" use case.

Figure 79 : Design: Tick Object / Game Logic System (Component Sequence)

Design: Tick Object / Game Logic System (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick Object System	System (Ticked)	Object & Object Management System (Data)	This is equivalent to telling the game to perform game logic.
2	//Tick Commander Objects	Object & Object Management	Object & Object Management	Tell the commanders to perform general game logic for the units it commands.

		System (Data)	System (Data)	
3	//Tick Unit Objects	Object & Object Management System (Data)	Object & Object Management System (Data)	Perform Game logic on the individual units themselves.

A - 1.1.1.3.1.4.6.1.2 Update Commander Object

Type: *public* **UseCase**
Package: Tick Object Component

This use case represents the object performing the game logic relavant to the object.

The commander is responsible for performing the general strategy for the computer player.

Scenarios

Basic Path {Basic Path}.
DESCRIPTION:

1. Update position based on speed and movement dir.
2. Perform action depending on state:
   - If an object is in an ATTACKING state, it would fire it's weapon.
   - etc.

A - 1.1.1.3.1.4.6.1.3 Update Controlled Object

Type: *public* **UseCase**
Package: Tick Object Component

Represents performing game logic for an individual unit in the game.

A - 1.1.1.3.1.4.7 Tick UI Component

The diagram shows the use cases involved in the ticking of the UI component that is needed for the game of Starcraft(tm).

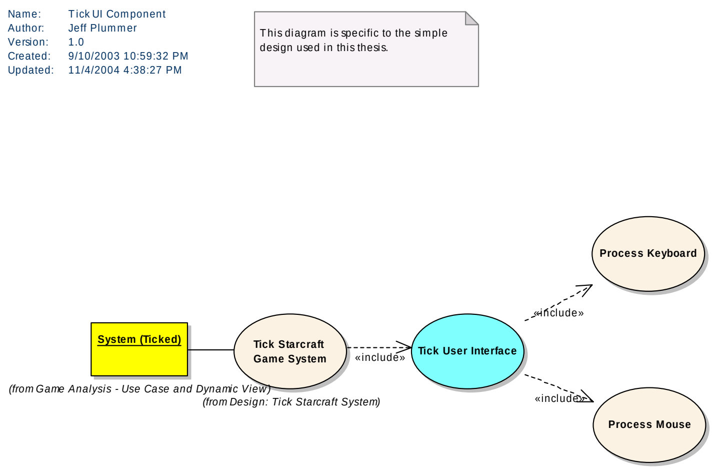


Figure 80 : Tick UI Component

A - 1.1.1.3.1.4.7.1.1 Process Keyboard

Type: *public* **UseCase**
Package: Tick UI Component

A - 1.1.1.3.1.4.7.1.2 Process Mouse

Type: *public* **UseCase**
Package: Tick UI Component

A - 1.1.1.3.1.4.7.1.3 Tick User Interface

Type: *public* **UseCase**
Package: Tick UI Component

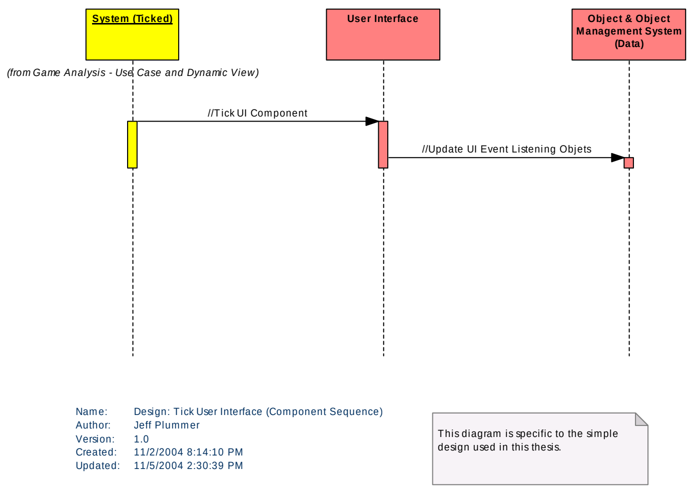
Tick the User Interface Component. Reads and processes all forms of input.

Starcraft UI system will read mouse movements and mouse clicks.

Scenarios

Tick User Interface {Basic Path}.

1. Read captured keyboard/mouse events
2. Request views of keyboard/mouse listener objects.
3. Update game logic keyboard/mouse listener objects.



This diagram shows the sequence of events at the component level that occur to complete the "Tick User Interface System" use case.

Figure 81 : Design: Tick User Interface (Component Sequence)

Design: Tick User Interface (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick UI Component	System (Ticked)	User Interface	While the UI event collection may occur in a seperate thread, the main thread (tick) will takes those events and process them.
2	//Update UI Event	User Interface	Object & Object	Update the objects that are responsible for receiving UI events.

	Listenin g Objets		Manage ment System (Data)	When these UI Listening objects get ticked during the "tick object component" game logic will do something based on the UI events.

A - 1.1.1.4 Unreal Tournament

This package represents the analysis and design work performed for the game Unreal Tournament(tm).

A - 1.1.1.4.1 Use Cases

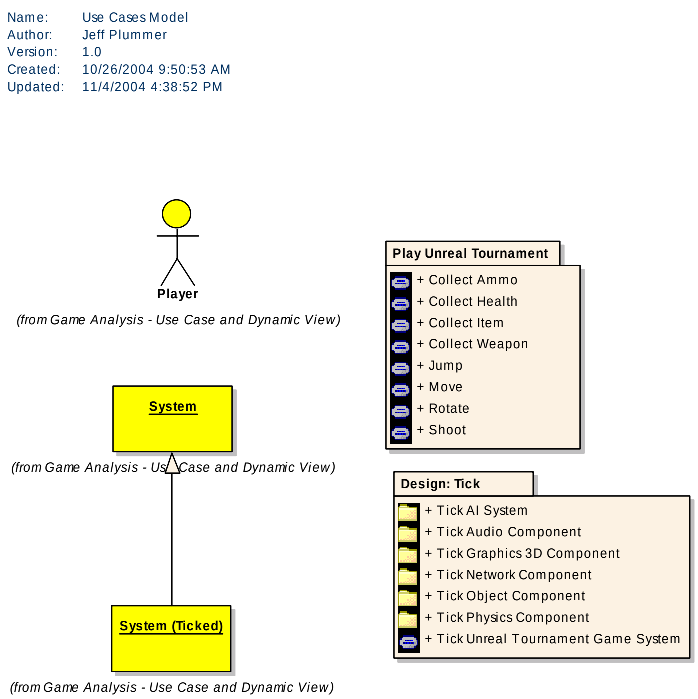


Figure 82 : Use Cases Model

A - 1.1.1.4.1.1 Play Unreal Tournament

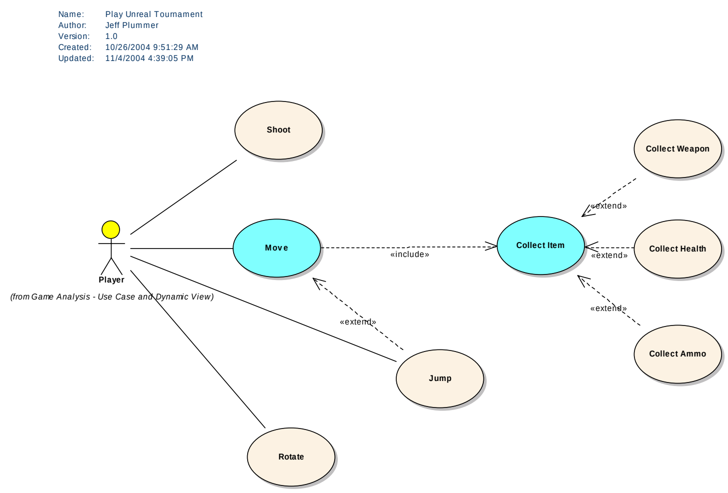


Figure 83 : Play Unreal Tournament

A - 1.1.1.4.1.1.1.1.1 Collect Ammo

Type: *public* **UseCase**
Package: Play Unreal Tournament

A - 1.1.1.4.1.1.1.1.2 Collect Health

Type: *public* **UseCase**
Package: Play Unreal Tournament

A - 1.1.1.4.1.1.1.1.3 Collect Item

Type: *public* **UseCase**
Package: Play Unreal Tournament

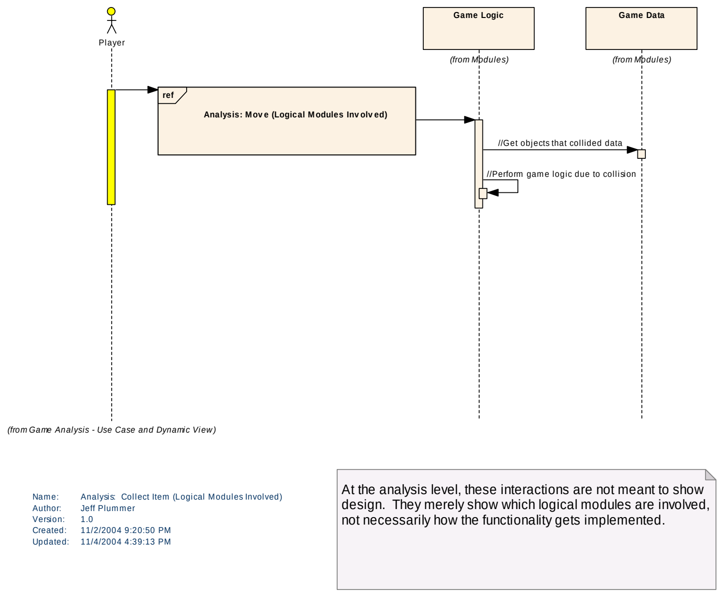


Figure 84 : Analysis: Collect Item (Logical Modules Involved)

Analysis: Collect Item (Logical Modules Involved) Messages

ID	Message	From Object	To Object	Notes
1		Player	Analysis: Move (Logical Modules Involved)	
2		Analysis: Move (Logical Modules Involved)	Game Logic	A movement results in a player's collision with an "item" game object. Could be a weapon, or health, etc.
3	//Get objects that	Game Logic	Game Data	

	collided data			
4	//Perform game logic due to collision	Game Logic	Game Logic	Could be to increase player health, add ammo, etc.

A - 1.1.1.4.1.1.1.1.4 Collect Weapon

Type: *public* **UseCase**
Package: Play Unreal Tournament

A - 1.1.1.4.1.1.1.1.5 Jump

Type: *public* **UseCase**
Package: Play Unreal Tournament

A - 1.1.1.4.1.1.1.1.6 Move

Type: *public* **UseCase**
Package: Play Unreal Tournament

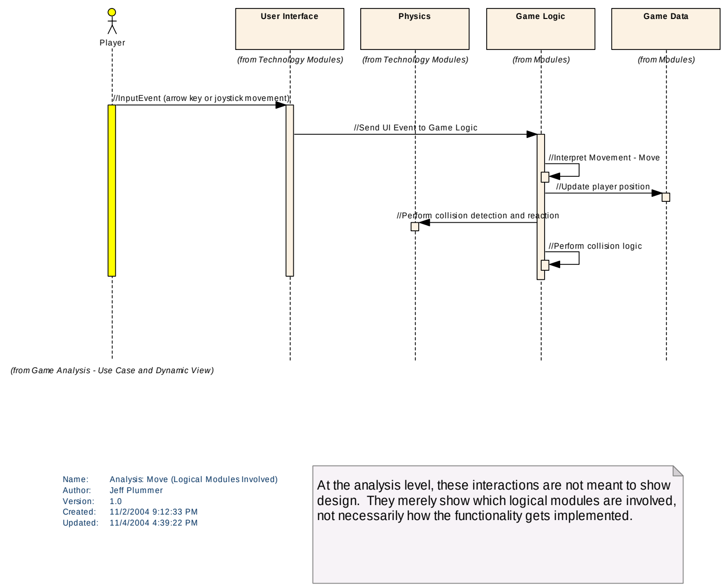


Figure 85 : Analysis: Move (Logical Modules Involved)

Analysis: Move (Logical Modules Involved) Messages

ID	Message	From Object	To Object	Notes
1	//InputEvent (arrow key or joystick movement)	Player	User Interface	Player presses the an arrow movement key or moves the joystick signaling an event.
2	//Send UI Event to Game Logic	User Interface	Game Logic	The game logic will determine how to interpret the UI event.
3	//Interpret Movement - Move	Game Logic	Game Logic	The game logic determines that the players position needs to be moved based on the input command.

4	//Update player position	Game Logic	Game Data	Update the player's position data.
5	//Perform collision detection and reaction	Game Logic	Physics	Its an arbitrary decision to say that the physics logical module performs the collision detection. This seems to be the trend in commercial physics engines so I'm just continuing the trend.
6	//Perform collision logic	Game Logic	Game Logic	When you collide with certain objects (ammo, health, etc) game logic needs to get involved.

A - 1.1.1.4.1.1.1.1.7 Rotate

Type: *public* **UseCase**
Package: Play Unreal Tournament

A - 1.1.1.4.1.1.1.1.8 Shoot

Type: *public* **UseCase**
Package: Play Unreal Tournament

A - 1.1.1.4.1.2 Design: Tick

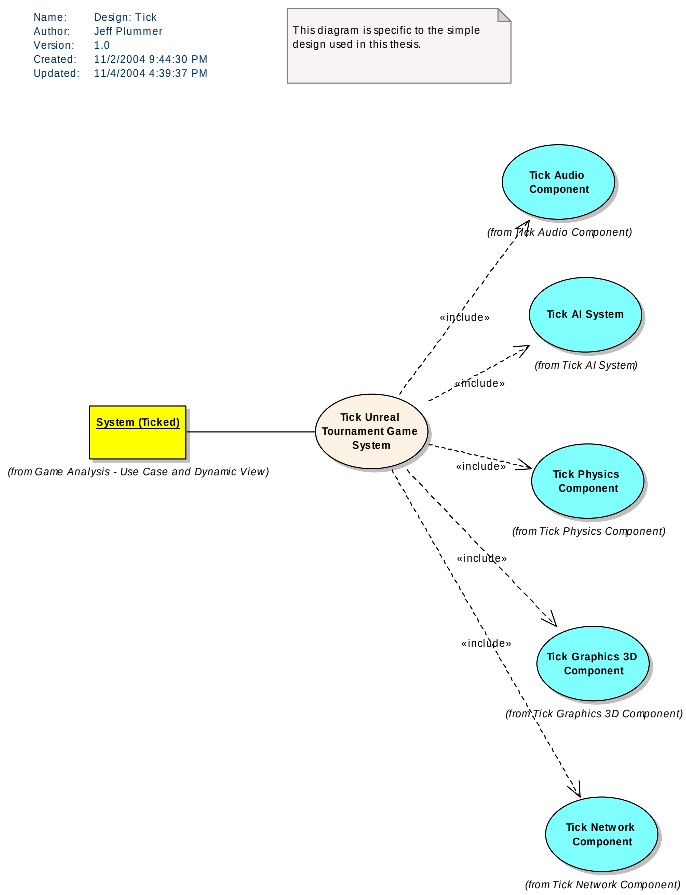


Figure 86 : Design: Tick

A - 1.1.1.4.1.2.1.1.1 System (Ticked)

Type: *public* **Object**
Package: Game Analysis - Use Case and Dynamic View

This actor represents the System but implies the actions occur on a regular or clocked basis.

A - 1.1.1.4.1.2.1.1.2 Tick Physics Component

Type: *public* **UseCase**
Package: Tick Physics Component

A - 1.1.1.4.1.2.1.1.3 Tick AI System

Type: *public* **UseCase**
Package: Tick AI System

Tick the artificial intelligence component. Execute AI operations that determine what the the objects intend to do next.

Starcraft AI system will determine computer players' decisions, an object's next move, and some AI state information.

Scenarios

Tick AI System {Basic Path}.

1. Request views/object lists of AI objects to process.
2. Read object AI related information (state, etc.).
3. Process objects.

NOTE: Objects don't exist in a vacuum. The AI system could provide messaging, etc. for AI interactions to take place between objects.

A - 1.1.1.4.1.2.1.1.4 Tick Audio Component

Type: *public* **UseCase**
Package: Tick Audio Component

A - 1.1.1.4.1.2.1.1.5 Tick Graphics 3D Component

Type: *public* **UseCase**
Package: Tick Graphics 3D Component

A - 1.1.1.4.1.2.1.1.6 Note

Type: *public* **Note**
Package: Tick Graphics 3D Component

This diagram is specific to the simple design used in this thesis.

A - 1.1.1.4.1.2.1.1.7 Tick Network Component

Type: *public* **UseCase**
Package: Tick Network Component

A - 1.1.1.4.1.2.1.1.8 Tick Unreal Tournament Game System

Type: *public* **UseCase**
Package: Design: Tick

A - 1.1.1.4.1.2.2 Tick AI System

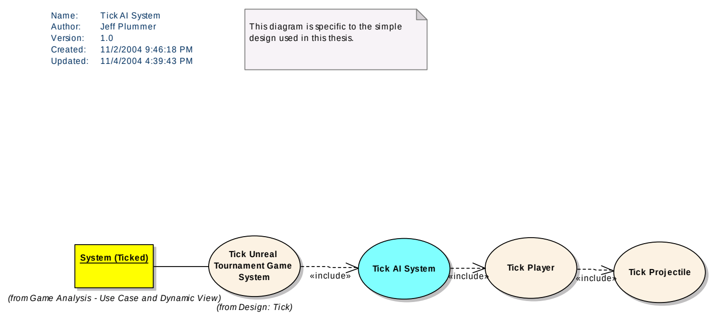


Figure 87 : Tick AI System

A - 1.1.1.4.1.2.2.1.1 Tick Unreal Tournament Game System

Type: *public* **UseCase**
Package: Design: Tick

A - 1.1.1.4.1.2.2.1.2 System (Ticked)

Type: *public* **Object**
Package: Game Analysis - Use Case and Dynamic View

This actor represents the System but implies the actions occur on a regular or clocked basis.

A - 1.1.1.4.1.2.2.1.3 Note

Type: *public* **Note**
Package: Tick Graphics 3D Component

This diagram is specific to the simple design used in this thesis.

A - 1.1.1.4.1.2.2.1.4 Tick AI System

Type: *public* **UseCase**

Package: Tick AI System

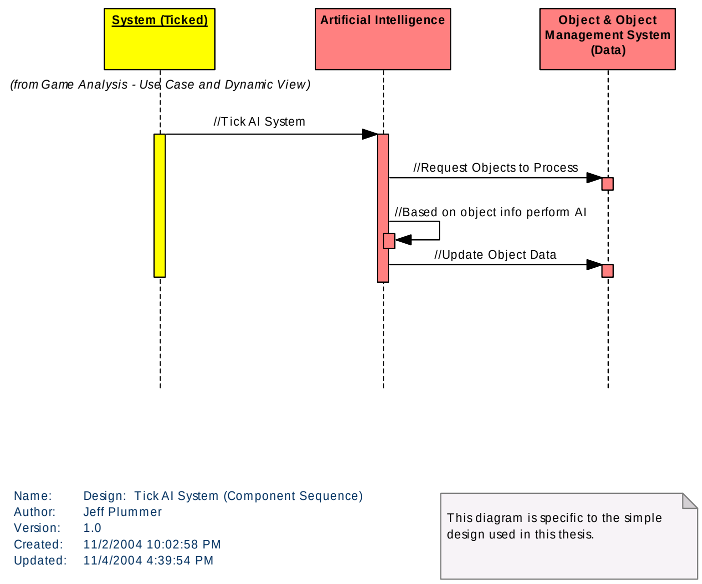


Figure 88 : Design: Tick AI System (Component Sequence)

Design: Tick AI System (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick AI System	System (Ticked)	Artificial Intelligence	In this design the AI resides in its own component and will be "ticked" to tell the AI system to operate on a list of objects.
2	//Request Objects to Process	Artificial Intelligence	Object & Object Management System	Request list(s) of objects that require AI processing.

			(Data)	
3	//Based on object info perform AI	Artificial Intelligence	Artificial Intelligence	Objects are like state machines and depending on their state, different types AI processing will be done for each object.
4	//Update Object Data	Artificial Intelligence	Object & Object Management System (Data)	After AI object processing has completed, update the object data.

A - 1.1.1.4.1.2.2.1.5 Tick Player

Type: *public* **UseCase**
Package: Tick AI System

Tick each computer and human controlled player object.

A - 1.1.1.4.1.2.2.1.6 Tick Projectile

Type: *public* **UseCase**
Package: Tick AI System

This is an arbitrary decision, we are saying as part of ticking the player, the player will tick all the projectiles the player has created. Basically this is simply for network player functionality distribution.

A - 1.1.1.4.1.2.3 Tick Audio Component

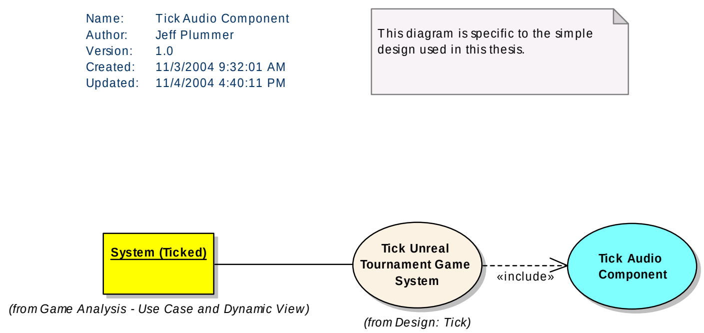


Figure 89 : Tick Audio Component

A - 1.1.1.4.1.2.3.1.1 Tick Audio Component

Type: *public* **UseCase**

Package: Tick Audio Component

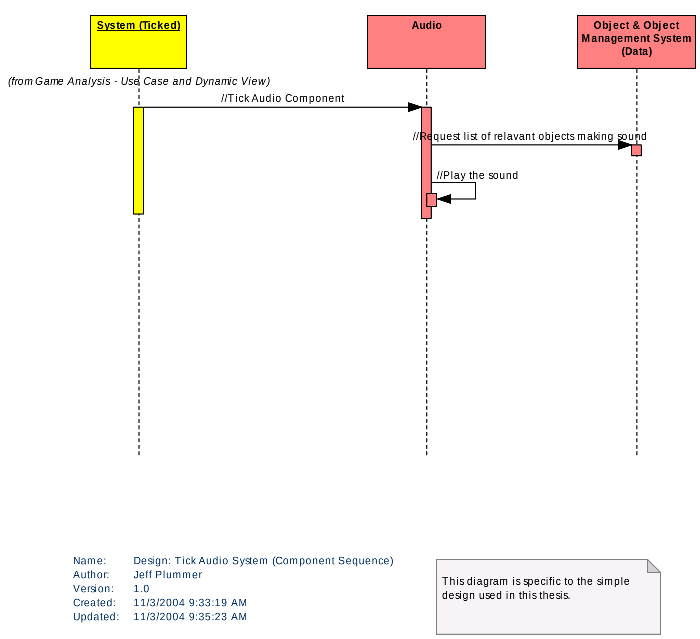


Figure 90 : Design: Tick Audio System (Component Sequence)

Design: Tick Audio System (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick Audio Component	System (Ticked)	Audio	In this design the Audio resides in its own component and will be "ticked" to tell the Audio system to operate on a list of objects.
2	//Request list of relavant objects making sound	Audio	Object & Object Management System (Data)	Request list of objects near the "camera" that are currently making sounds.
3	//Play the sound	Audio	Audio	Send the sounds to the sound card

A - 1.1.1.4.1.2.4 Tick Graphics 3D Component

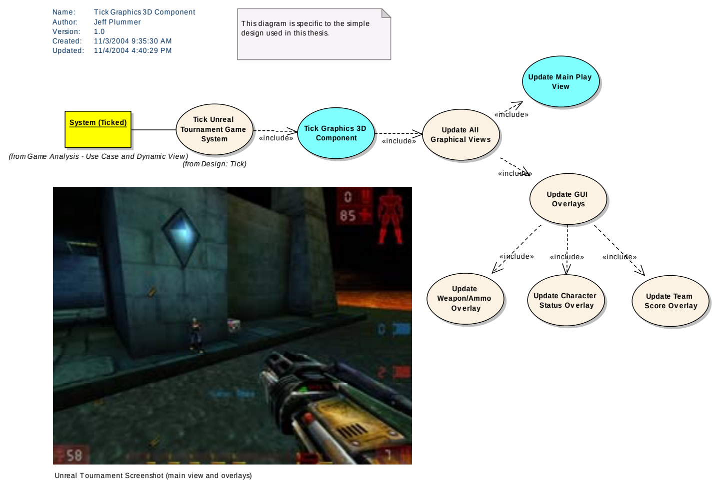


Figure 91 : Tick Graphics 3D Component

A - 1.1.1.4.1.2.4.1.1 <u>Tick Graphics 3D Component</u>

Type: *public* **<u>UseCase</u>**
Package: Tick Graphics 3D Component

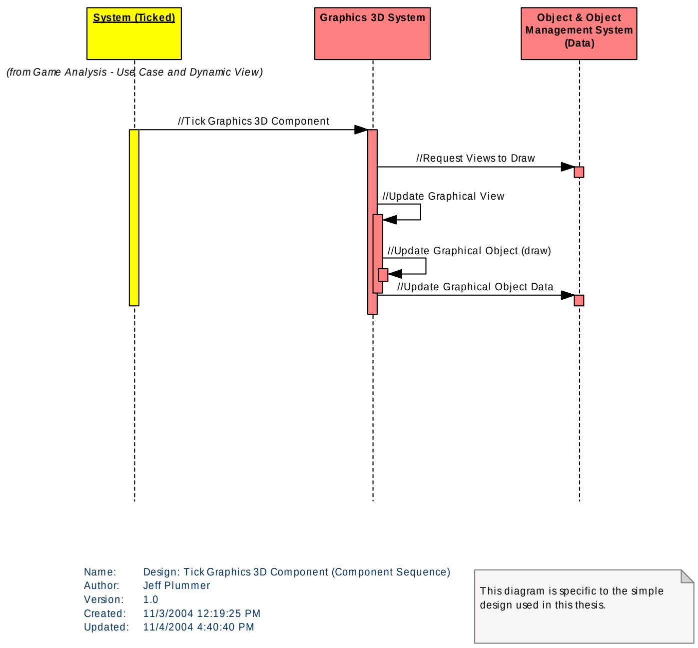


Figure 92 : Design: Tick Graphics 3D Component (Component Sequence)

Design: Tick Graphics 3D Component (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick Graphics 3D Component	System (Ticked)	Graphics 3D System	Ticking the graphics 3D component causes all visible views and objects to be drawn.
2	//Request Views to Draw	Graphics 3D System	Object & Object Management System (Data)	Request views (context and object list) to draw.
3	//Update Graphical View	Graphics 3D System	Graphics 3D System	For each view...

4	//Update Graphical Object (draw)	Graphics 3D System	Graphics 3D System	Draw each graphical object using the view context.
5	//Update Graphical Object Data	Graphics 3D System	Object & Object Management System (Data)	Update an necessary data like screen coords that may be used by other systems.

A - 1.1.1.4.1.2.4.1.2 Update All Graphical Views

Type: *public* **UseCase**
Package: Tick Graphics 3D Component

A - 1.1.1.4.1.2.4.1.3 Update Character Status Overlay

Type: *public* **UseCase**
Package: Tick Graphics 3D Component

A - 1.1.1.4.1.2.4.1.4 Update GUI Overlays

Type: *public* **UseCase**
Package: Tick Graphics 3D Component

A - 1.1.1.4.1.2.4.1.5 Update Main Play View

Type: *public* **UseCase**
Package: Tick Graphics 3D Component

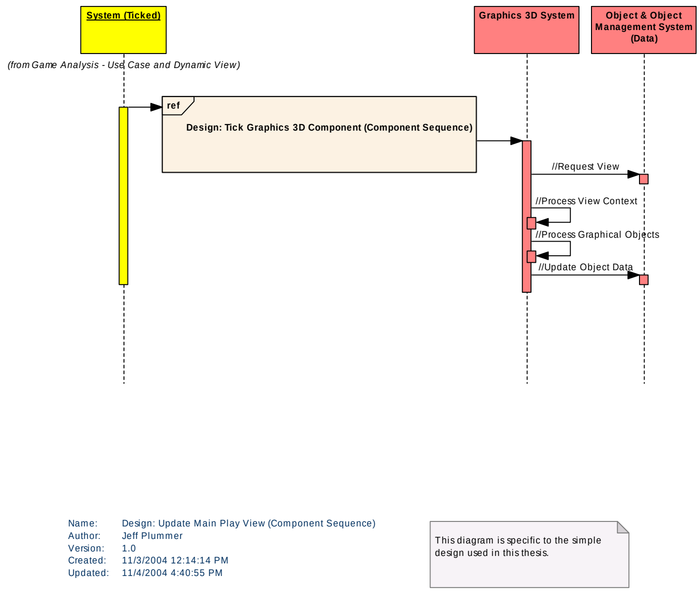


Figure 93 : Design: Update Main Play View (Component Sequence)

Design: Update Main Play View (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1		System (Ticked)	Design: Tick Graphics 3D Component (Component Sequence)	
2		Design: Tick Graphics 3D Component (Compo	Graphics 3D System	During the course of ticking the graphics 3D component we come to this functionality of updating the graphical view.

		nent Sequence)		
3	//Request View	Graphics 3D System	Object & Object Management System (Data)	Request the view to draw from the object system.
4	//Process View Context	Graphics 3D System	Graphics 3D System	Understand things like view size on screen, coordinate system, camera location, etc.
5	//Process Graphical Objects	Graphics 3D System	Graphics 3D System	Draw the objects in the object list of the view using the view context.
6	//Update Object Data	Graphics 3D System	Object & Object Management System (Data)	Update things like screen coords. data, etc. in case other components require that data.

A - 1.1.1.4.1.2.4.1.6 Update Team Score Overlay

Type: *public* **UseCase**
Package: Tick Graphics 3D Component

A - 1.1.1.4.1.2.4.1.7 Update Weapon/Ammo Overlay

Type: *public* **UseCase**
Package: Tick Graphics 3D Component

A - 1.1.1.4.1.2.5 Tick Network Component

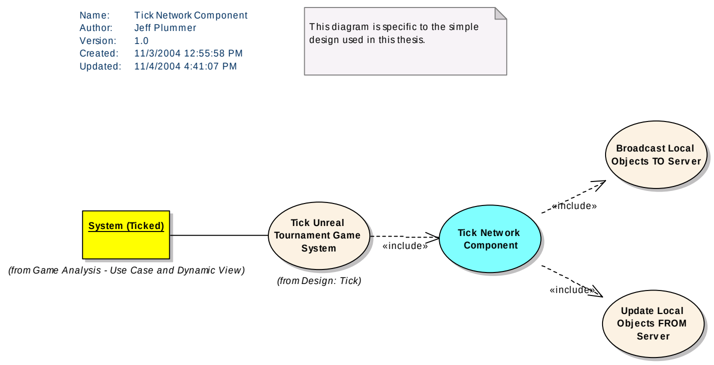


Figure 94 : Tick Network Component

A - 1.1.1.4.1.2.5.1.1 Broadcast Local Objects TO Server

Type: *public* **UseCase**
Package: Tick Network Component

A - 1.1.1.4.1.2.5.1.2 Tick Network Component

Type: *public* **UseCase**
Package: Tick Network Component

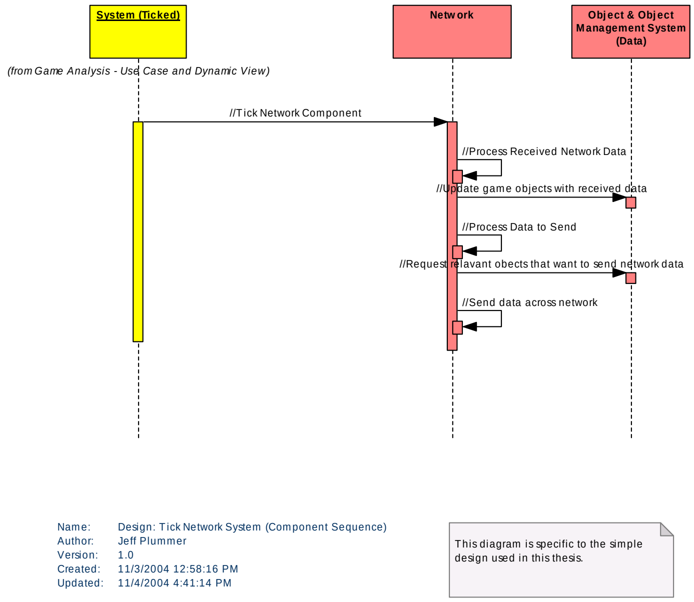


Figure 95 : Design: Tick Network System (Component Sequence)

Design: Tick Network System (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick Network Component	System (Ticked)	Network	In this design the network resides in its own component and will be "ticked" to tell the network system to operate on a list of objects.
2	//Process Received Network Data	Network	Network	The network receiving of network data is in it's own thread, but act of doing something meaningful with the network data is performed in the main tick.
3	//Update game	Network	Object &	Update the proper local objects with the updates

	objects with receive d data		Object Manage ment System (Data)	that came from the network.
4	//Proces s Data to Send	Networ k	Networ k	
5	//Reque st relavant obects that want to send network data	Networ k	Object & Object Manage ment System (Data)	The object managment system will determine which objects are relavant to networked computers and should send their network data.
6	//Send data across network	Networ k	Networ k	Send the proper data across the network.

A - 1.1.1.4.1.2.5.1.3 Update Local Objects FROM Server

Type: *public* **UseCase**
Package: Tick Network Component

A - 1.1.1.4.1.2.6 Tick Object Component

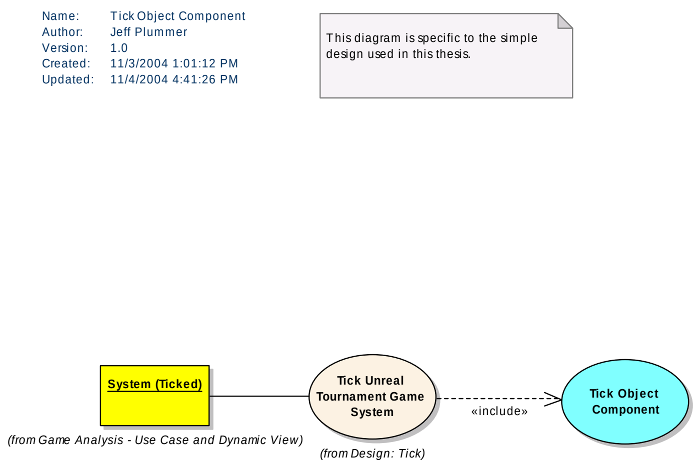


Figure 96 : Tick Object Component

A - 1.1.1.4.1.2.6.1.1 Tick Object Component

Type: *public* **UseCase**
Package: Tick Object Component

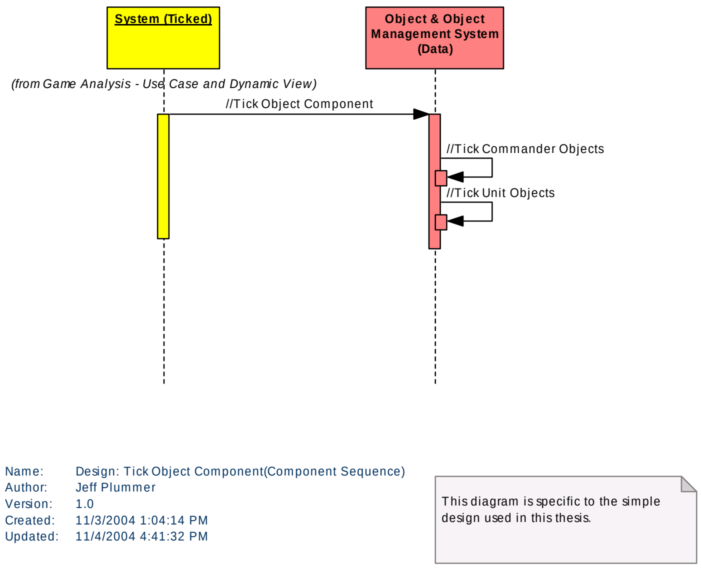


Figure 97 : Design: Tick Object Component(Component Sequence)

Design: Tick Object Component(Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick Object Component	System (Ticked)	Object & Object Management System (Data)	This is equivalent to telling the game to perform game logic.
2	//Tick Commander Objects	Object & Object Management System (Data)	Object & Object Management System (Data)	Tell the commanders to perform general game logic for the units it commands.
3	//Tick Unit Objects	Object & Object Manage	Object & Object Manage	Perform Game logic on the individual units themselves.

		ment System (Data)	ment System (Data)	

A - 1.1.1.4.1.2.7 Tick Physics Component

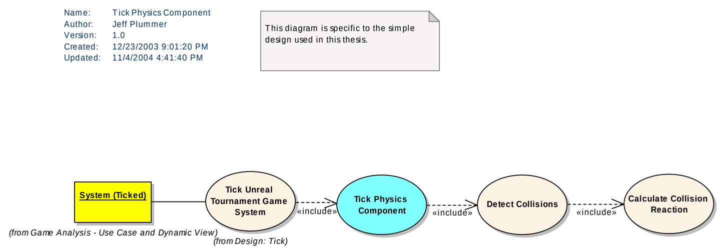


Figure 98 : Tick Physics Component

A - 1.1.1.4.1.2.7.1.1 Calculate Collision Reaction

Type: *public* **UseCase**
Package: Tick Physics Component

Upon a collision, calculate the physical reaction that occurs (i.e. bounce).

A - 1.1.1.4.1.2.7.1.2 Detect Collisions

Type: *public* **UseCase**
Package: Tick Physics Component

This is an arbitrary decision but we are saying collsion detection functionality resides in the physics engine. Many commercial physics engines offer this functionality, and I'm just continuing that.

A - 1.1.1.4.1.2.7.1.3 Tick Physics Component

Type: *public* **UseCase**
Package: Tick Physics Component

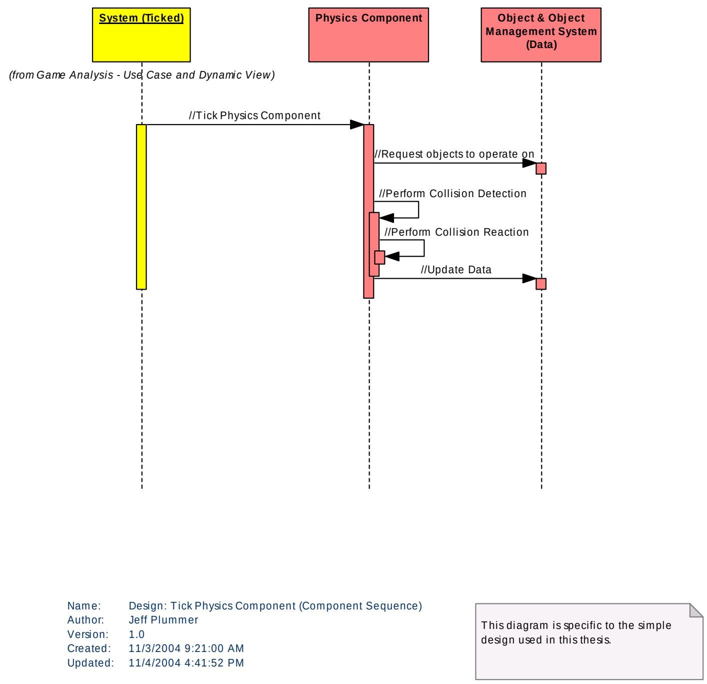


Figure 99 : Design: Tick Physics Component (Component Sequence)

Design: Tick Physics Component (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick Physics Component	System (Ticked)	Physics Component	In this design the physics resides in its own component and will be "ticked" to tell the physics system to operate on a list of objects.
2	//Request objects to operate on	Physics Component	Object & Object Management System (Data)	Request list(s) of objects that require physics processing. Basically request only the objects to perform collision detection and reaction on.
3	//Perform	Physics Compo	Physics Compo	Determine if objects collide

	Collision Detection	nent	nent	
4	//Perform Collision Reaction	Physics Component	Physics Component	Determine the physical reaction that occurs due to the collision(s)
5	//Update Data	Physics Component	Object & Object Management System (Data)	Update object data based on the physical reaction.

APPENDIX B –
PROTOTYPE DESIGN

TABLE OF CONTENTS – APPENDIX B

SECTION NAME Page

Prototype 218

Analysis View 218

Logical Architecture 218

Object Interfaces 219

GameObject 219

AI2Object 220

IAIObject 220

IGraphics2DObject 220

IGraphics3DObject 221

Logical View 222

Programming Utilities Library 222

Systems 223

AI System 224

AI Component - Implementation 224

AI Exported Classes 225

Root 225

Private AI System Implementation 227

CAISystem 227

CAIProcessorObject 228

CAIViewProcessor 229

AI Component - Interfaces 231

Section Name Page

AI Interfaces Object System Can Use To Communicate With AI System 232

IAIProcessorObject 232

IAISystem 232

IAIViewProcessor 233

AI Interfaces The Object System Implements 234

IAICapableObject 234

IAIObjectSystem 234

IAIProcessableObject 235

IAISceneManager 236

IAIView 236

AI2System 238

AI2 Component - Implementation 238

AI2 Exported Classes 239

Root 239

Private AI2 System Implementation 241

CAI2System 241

CAI2ProcessorObject 242

CAI2ViewProcessor 243

AI2 Component - Interfaces 245

AI2 Interfaces Object System Can Use To Communicate With AI2 System 246

IAI2ProcessorObject 246

IAI2System 246

Section Name Page

IAI2ViewProcessor ... 247
AI2 Interfaces The Object System Implements ... 248
IAI2CapableObject ... 248
IAI2ObjectSystem ... 248
IAI2ProcessableObject ... 249
IAI2SceneManager ... 250
IAI2View ... 250
Game Object System ... 252
Game Object Component - Implementation ... 252
Game Object Component Exported Classes ... 252
Root ... 252
Private Game Object Component Implementation ... 254
CDemoCamera ... 254
CDemoGameObjectSystem ... 255
CDemoMainView ... 259
CDemoObject ... 259
CDemoObjectSceneManager ... 265
CDemoViewBaseClass ... 267
CTriangleGameObject ... 273
Data Structures ... 275
demoPoint2i ... 275
demoPoint3f ... 276

Section Name Page

demoRect 276

Game Object Component - Interfaces 278

IObjectSystem 278

Component Attachings 279

Game System 281

CDemoApplication 281

Graphic 3D System 284

Graphics3DComponent - Implementation 284

Exported Classes 285

Root 285

Private Graphics3D System Implementation 287

CGraphics3DProcessorObject 287

CGraphics3DSystem 288

CGraphics3DViewProcessor 291

Graphics3DComponent - Interfaces 293

Interfaces the Object System can use to communicate with the Graphics3D System 294

IGraphics3DProcessorObject 294

IGraphics3DSystem 295

IGraphics3DViewProcessor 295

Interfaces The Object System Implements 297

IGraphics3DCamera 297

Section Name Page

IGraphics3DCapableObject 297

IGraphics3DObjectSystem 298

IGraphics3DProcessableObject 298

IGraphics3DSceneManager 299

IGraphics3DView 300

Graphics 2D System 302

Graphics Component - Implementation 302

Exported Classes 303

Root 303

Private Graphics System Implementation 305

CGraphicsProcessorObject 305

CGraphicsSystem 308

CGraphicsViewProcessor 310

Graphics Component - Interfaces 312

Interfaces Object System Can Use To Communicate With Graphics System 313

IGraphicsProcessorObject 313

IGraphicsSystem 313

Interfaces The Object System Implements 315

I2DGraphicsCamera 315

I2DGraphicsObject 316

I2DSpriteGraphicsObject 316

IGraphicsCamera 317

Section Name Page

IGraphicsCapableObject.......... 317
IGraphicsObjectIterator.......... 317
IGraphicsObjectSystem.......... 318
IGraphicsSceneManager.......... 318
IGraphicsView.......... 319
IGraphicsViewIterator.......... 320
IProcessableGraphicsObject.......... 321
Utility Includes.......... 323
CStdStr.......... 323
IIterator.......... 334
VectorBasedIteratorTemplateClass.......... 335
Dynamic View.......... 337
Initialize.......... 337
Initialize AI2 System.......... 337
Initialize AI System.......... 340
Initialize Graphics 3D System.......... 343
Initialize Graphics System.......... 346
Initialize Object System.......... 350
Initialize Game System.......... 353
Tick.......... 356
Tick AI System.......... 356
Tick AI2 System.......... 362

Section Name Page

Tick Graphics 3D System 367

Tick Graphics System 373

Tick Prototype Game System 379

Component View 380

AI System 2 380

Artificial Intelligence 380

Audio 380

Game System 380

Graphics 381

Graphics 3D System 381

Network 381

Object & Object Management System (Data) 381

OGRE Graphics Engine 381

Physics Component 382

User Interface 382

B - 1.2 Prototype

B - 1.2.1 Analysis View

This view shows a quick analysis of what the prototype is.

B - 1.2.1.1 Logical Architecture

This diagram shows the high level architecture of the prototype system that was built.

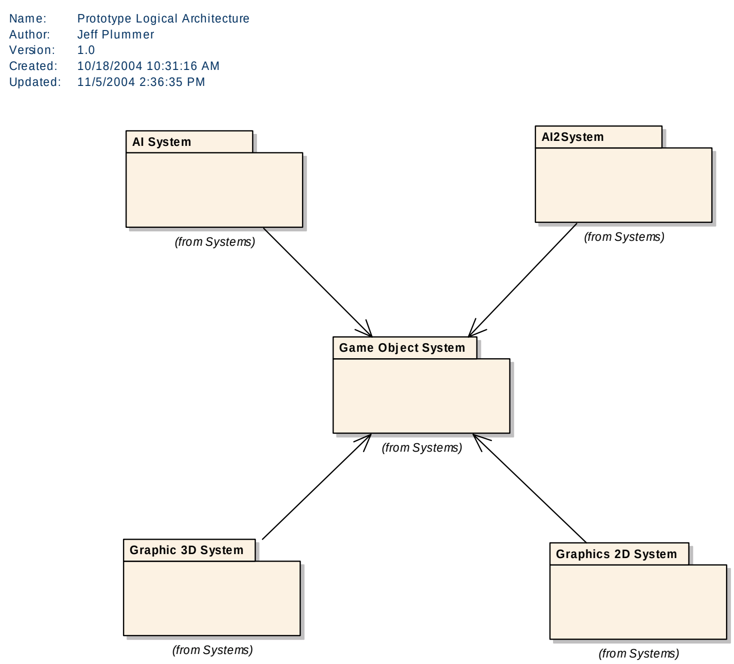


Figure 100 : Prototype Logical Architecture

B - 1.2.1.1.1 Object Interfaces

This diagram shows a short list of data that will reside in the prototype "game" object, and who will use that data.

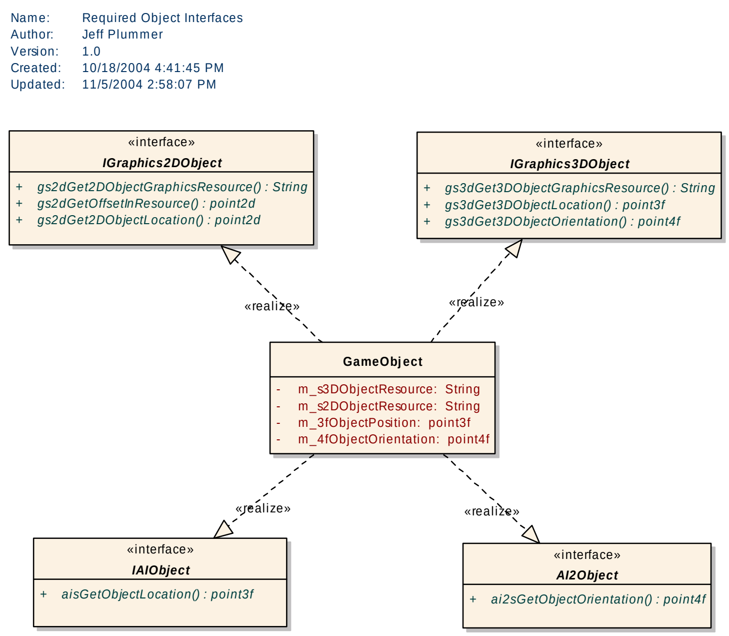


Figure 101 : Required Object Interfaces

B - 1.2.1.1.1.1.1.1.1 <u>GameObject</u>

Type: *public* **<u>Class</u>**
Implements: *AI2Object, IAIObject, IGraphics2DObject, IGraphics3DObject.*
Package: Object Interfaces

This example class shows what data will exist in a game object in the prototype.

GameObject Attributes

Attribute	Type	Notes
m_s3DObject Resource	private : *String*	A string that says what 3D graphical resource should be used to represent this object in 3D.
	private :	A string that says what 2D

m_s2DObject Resource	*String*	graphical resource should be used to represent this object in 2D.
m_3fObjectPo sition	private : *point3f*	The object's position in 3-Space.
m_4fObjectOr ientation	private : *point4f*	The object's orientation represented as a quaternion.

B - 1.2.1.1.1.1.1.1.2 AI2Object

Type: *public abstract «interface»* **Interface**
Package: Object Interfaces

This sample interface shows what type of data the AI2 engine will require from an AI2 object.

AI2Object Interfaces

Method	Type	Notes
ai2sGetObject Orientation ()	public abstract: *point4f*	Get the object's orientation represented as a quaternion.

B - 1.2.1.1.1.1.1.1.3 IAIObject

Type: *public abstract «interface»* **Interface**
Package: Object Interfaces

This sample interface shows what type of data the AI engine will require from an AI object.

IAIObject Interfaces

Method	Type	Notes
aisGetObjectL ocation ()	public abstract: *point3f*	Get the object's position in 3-Space.

B - 1.2.1.1.1.1.1.1.4 IGraphics2DObject

Type: *public abstract «interface»* **Interface**

Package: Object Interfaces

This sample interface shows what type of data the 2D Graphics engine will require from a 2D graphical object.

IGraphics2DObject Interfaces

Method	Type	Notes
gs2dGet2DObjectGraphicsResource ()	public abstract: *String*	A string that says what 2D graphical resource should be used to represent this object in 2D.
gs2dGetOffsetInResource ()	public abstract: *point2d*	Get the offset in the 2d image resource that respresents the sprite. Game logic in the game object will actually use the quaternion orientation and create the sprite image offset.
gs2dGet2DObjectLocation ()	public abstract: *point2d*	Get the position of the object in 2 space

B - 1.2.1.1.1.1.1.1.5 IGraphics3DObject

Type: *public abstract «interface»* **Interface**
Package: Object Interfaces

This sample interface shows what type of data the 3D Graphics engine will require from a 3D graphical object.

IGraphics3DObject Interfaces

Method	Type	Notes
gs3dGet3DObjectGraphicsResource ()	public abstract: *String*	Get A string that says what 3D graphical resource should be used to represent this object in 3D.
gs3dGet3DObjectLocation ()	public abstract: *point3f*	Get the object's position in 3-Space.
gs3dGet3DObjectOrientation ()	public abstract: *point4f*	Get the object's orientation represented as a quaternion.

B - 1.2.2 Logical View

This view shows the classes and structures involved in this prototype.

B - 1.2.2.1 Programming Utilities Library

This package contains many of the utility classes that were used in this project.

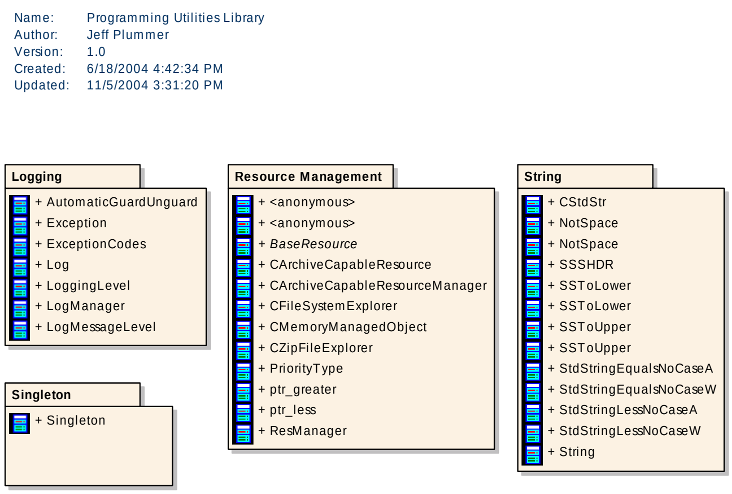


Figure 102 : Programming Utilities Library

B - 1.2.2.2 Systems

This package contains all the systems involved in the prototype.

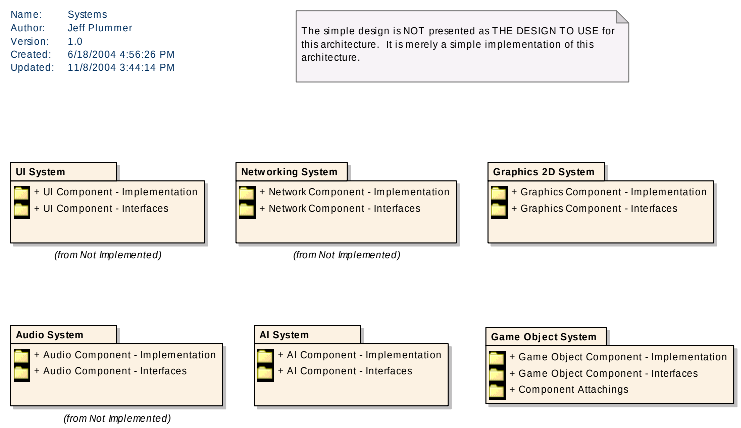


Figure 103 : Systems

B - 1.2.2.1.2 AI System

This represents one artifical intelligence logical module. It's functionality will be very simple, possibly adjust the object position in 3-space.

B - 1.2.2.1.1 AI Component - Implementation

This package contains an example implementation of the AI system. The implementation is not meant to show how to implement an AI engine, but rather show how an AI component could be built using the simple design presented in this thesis.

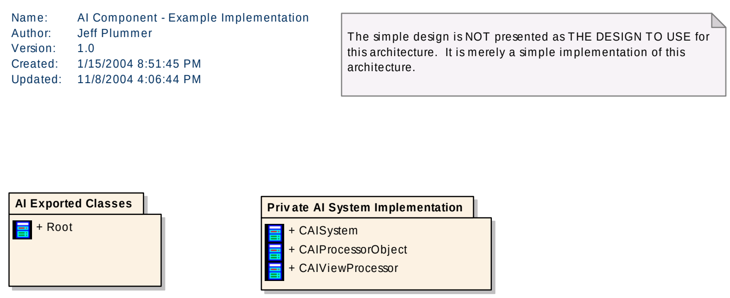


Figure 104 : AI Component - Example Implementation

B - 1.2.2.2.1.1.1 AI Exported Classes

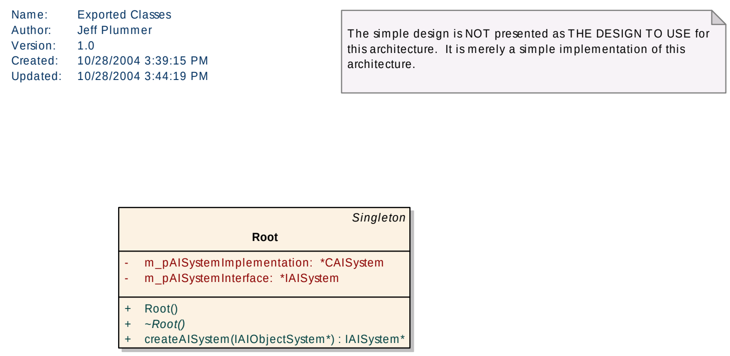


Figure 105 : Exported Classes

B - 1.2.2.2.1.1.1.1.1 <u>Root</u>

Type: *public* **<u>Class</u>**
Extends: *Singleton.*
Package: AI Exported Classes

This class is the only exported class in the Artifical Intelligence component. It represents the initial link to the AI system. From here the game system will connect to the AI system, and request an interface to the AI system. Root is not part of the formal architecture, it is an implementation connection point. In the real world it may be necessary to communicate in more ways with the logical component (due to specific library initializations, etc.). These "extra" communications can be done through the root object directly to the instance of the AI system, rather than through the architectural specified interface.

Root Attributes

Attribute	**Type**	**Notes**
m_pAISystem Implementatio n	private : *CAISyste m*	A pointer to the implementation of the AI system. This should never be accessed publicly. It exists to handle those special "real world" occasions where the architectural interface doesn't handle implementation specific features.

m_pAISystem Interface	private : *IAISystem*	This is a pointer to the architectural interface to the AI component. The creator of root will receive a pointer to this interface after calling "CreateAISystem".

Root Methods

Method	**Type**	**Notes**
Root ()	public:	Constructor - Create an instance of the AI system.
~Root ()	public abstract:	Destructor - Destroy the instance of the AI System.
createAISystem (*IAIObjectSystem**)	public: *IAISystem**	param: pObjectSystem [IAIObjectSystem* - inout] A pointer to an object that implements the IAIObjectSystem interface. The AI component will use this interface to communicate to the data section of the game system. Connect the object system to the AI system and return an interface to AI system @param pObjectSystem A pointer to an object that implements the IAIObjectSystem interface. The AI component will use this interface to communicate to the data section of the game system.

B - 1.2.2.2.1.1.2 Private AI System Implementation

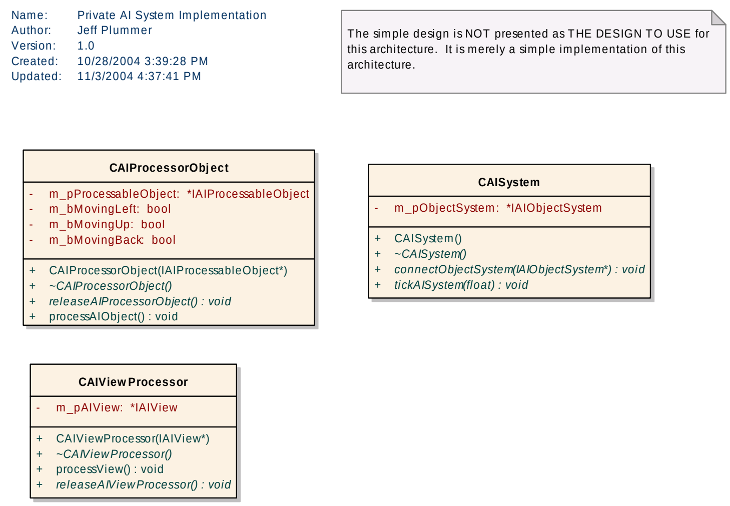


Figure 106 : Private AI System Implementation

B - 1.2.2.2.1.1.2.1.1 CAISystem

Type: *public* **Class**
Implements: *IAISystem.*
Package: Private AI System Implementation

This class represents the implementation of the AI system. It implements the IAISystem interface, and will be responsible for performing AI operations on the objects it receives from the Object component.

CAISystem Attributes

Attribute	Type	Notes
m_pObjectSystem	private : *IAIObjectSystem*	Pointer to the object system that this AI component is attached to.

CAISystem Methods

Method	Type	Notes
CAISystem ()	public:	Constructor
~CAISystem	public	Destructor

()	abstract:	
connectObject System (*IAIObjectSyst em**)	public abstract: *void*	param: objectSystem [IAIObjectSystem* - inout] A pointer to an object that implements the IAIObjectSystem interface. The AI component will use this interface to communicate to the data section of the game system. IAISystem interface implementation
tickAISystem (*float*)	public abstract: *void*	param: tDiff [float - in] IAISystem interface implementation Causes the AI component to iterate one cycle of time and performs AI processing on AI capable objects.

B - 1.2.2.2.1.1.2.1.2 CAIProcessorObject

Type: *public* **Class**
Implements: *IAIProcessorObject.*
Package: Private AI System Implementation

This is the AI Object observer that attaches to a game object. It uses the AI interface into the game object to get access to the necessary data. The AI Processor object will do that AI calculations treating the game object simply as a data access point.

CAIProcessorObject Attributes

Attribute	Type	Notes
m_pProcessabl eObject	private : *IAIProce ssableOb ject*	The AI Processable game object this observer is attached to.
m_bMovingLe ft	private : *bool*	AI variable used by the AI logic to determine the objects new position.
m_bMovingU p	private : *bool*	AI variable used by the AI logic to determine the objects new position.

m_bMovingBack	private : *bool*	AI variable used by the AI logic to determine the objects new position.

CAIProcessorObject Methods

Method	**Type**	**Notes**
CAIProcessorObject (*IAIProcessableObject**)	public:	param: pObject [IAIProcessableObject* - inout] Constructor
~CAIProcessorObject ()	public abstract:	Destructor
releaseAIProcessorObject ()	public abstract: *void*	The game object should call this function to delete the processor when the game object is deleted.
processAIObject ()	public: *void*	Perform AI Processing on the game object it is attached to. In this case just move the object around the screen.

B - 1.2.2.2.1.1.2.1.3 <u>CAIViewProcessor</u>

Type: *public* **<u>Class</u>**
Implements: *IAIViewProcessor.*
Package: Private AI System Implementation

This class attaches to a view and processes the view (i.e. uses the view interface to request objects and works with the object processors attached to the objects).

CAIViewProcessor Attributes

Attribute	**Type**	**Notes**
m_pAIView	private : *IAIView*	Pointer to the view being observed.

CAIViewProcessor Methods

Method	**Type**	**Notes**
CAIViewProcessor	public:	param: pView [IAIView* - inout]

(*IAIView**)		Constructor
~CAIViewProcessor ()	public abstract:	Destructor
processView ()	public: *void*	Perform AI Processing of this view. Request list of AI capable objects, and call their observer processors.
releaseAIViewProcessor ()	public abstract: *void*	Call during view destructor to release this observer.

B - 1.2.2.1.2 AI Component - Interfaces

This package contains an interfaces for the AI system. The interfaces presented here are for a specific design built on top of the proposed architecture.

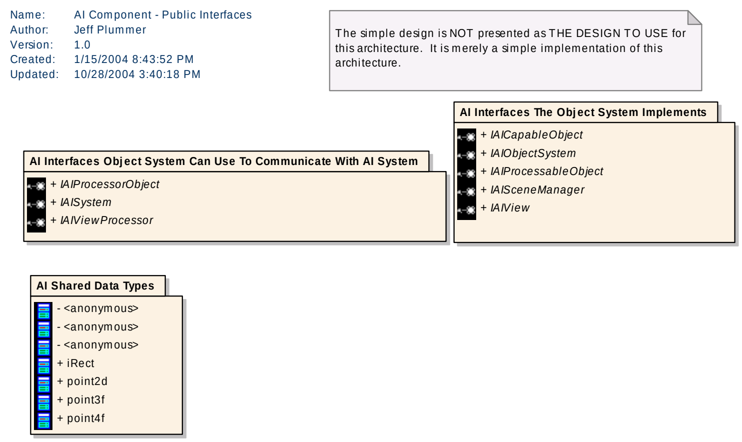


Figure 107 : AI Component - Public Interfaces

B - 1.2.2.2.1.2.1 AI Interfaces Object System Can Use To Communicate With AI System

This diagram shows the interfaces that are made available to the game system to use in order to communicate with the AI System.

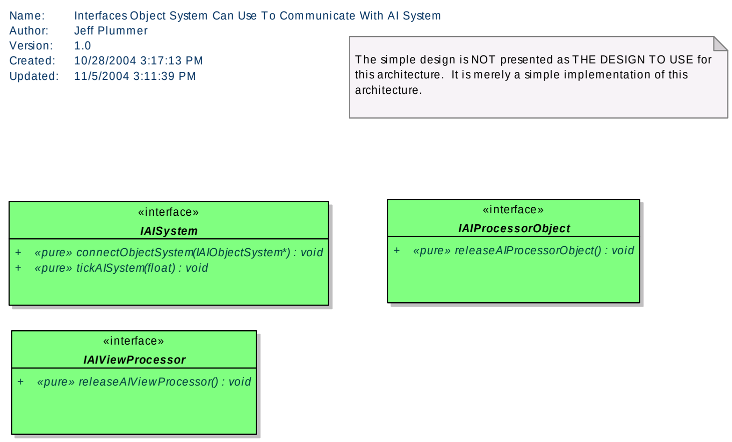


Figure 108 : Interfaces Object System Can Use To Communicate With AI System

B - 1.2.2.2.1.2.1.1.1 IAIProcessorObject

Type: *public abstract «interface»* **Interface**
Package: AI Interfaces Object System Can Use To Communicate With AI System

This is the interface the game system can use to access the domain-specific processor that is attached to a game object. This example is empty, showing that game objects don't necessarily require domain-specific functionality access.

IAIProcessorObject Interfaces

Method	Type	Notes
releaseAIProcessorObject ()	«pure» public abstract: *void*	Only required in C++ because there is no memory management. Call this during the game object destructor.

B - 1.2.2.2.1.2.1.1.2 IAISystem

Type: *public abstract «interface»* **Interface**
Package: AI Interfaces Object System Can Use To Communicate With AI System

This interface is the architectural connection from the game system to the AI component. One of the major goals of this architecture is to limit interaction from outside into the AI component. So this interface will provide only the functionality to setup the AI system and provide the AI system with the means to communicate back to the data. From that point on most communication will originate from the AI system back to the data.

IAISystem Interfaces

Method	Type	Notes
connectObject System (*IAIObjectSyst em**)	«pure» public abstract: *void*	param: objectSystem [IAIObjectSystem* - inout] Use this method to connect an AI Capable Object Management System to the AI Component.
tickAISystem (*float*)	«pure» public abstract: *void*	param: tDiff [float - in] Use this method to Tick the AI system, so that it will request and process AI objects.

B - 1.2.2.2.1.2.1.1.3 IAIViewProcessor

Type: *public abstract «interface»* **Interface**
Package: AI Interfaces Object System Can Use To Communicate With AI System

This is the interface the game system can use to access the domain-specific view processor that is attached to a view. This example is empty, showing that game views don't necessarily require domain-specific functionality access.

IAIViewProcessor Interfaces

Method	Type	Notes
releaseAIView Processor ()	«pure» public abstract: *void*	Only required in C++ because there is no memory management. Call this during the game object destructor.

B - 1.2.2.2.1.2.2 AI Interfaces The Object System Implements

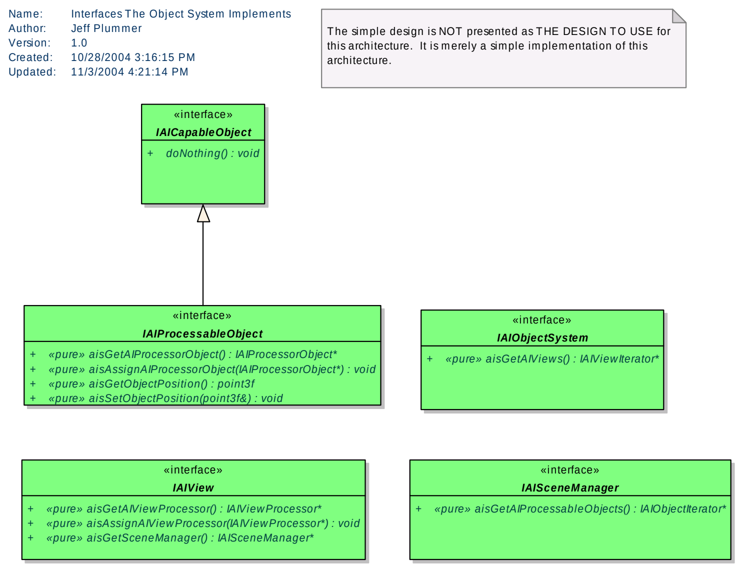


Figure 109 : Interfaces The Object System Implements

B - 1.2.2.2.1.2.2.1.1 <u>IAICapableObject</u>

Type: *public abstract «interface»* **<u>Interface</u>**
Package: AI Interfaces The Object System Implements

This class is required for C++ and dynamic type casting. It has no other uses.

IAICapableObject Interfaces

Method	Type	Notes
doNothing ()	public abstract: *void*	

B - 1.2.2.2.1.2.2.1.2 <u>IAIObjectSystem</u>

Type: *public abstract «interface»* **<u>Interface</u>**
Package: AI Interfaces The Object System Implements

This interface is the architectural connection from the object system responsible for managing objects capable of AI to the AI component. Using this interface the AI component will request AI capable objects and perform the appropriate AI operations on them.

IAIObjectSystem Interfaces

Method	Type	Notes
aisGetAIViews ()	«pure» public abstract: *IAIViewIterator**	Get an iterator (list) of active views to process.

B - 1.2.2.2.1.2.2.1.3 IAIProcessableObject

Type: *public abstract «interface»* **Interface**
Extends: *IAICapableObject.*
Package: AI Interfaces The Object System Implements

Game objects that wish to be processable by this AI engine must implement this interface. It allows the AI system to read/write certain data elements.

IAIProcessableObject Interfaces

Method	Type	Notes
aisGetAIProcessorObject ()	«pure» public abstract: *IAIProcessorObject**	Allows the AI engine to get the AI observer object attached to this game object.
aisAssignAIProcessorObject (*IAIProcessorObject**)	«pure» public abstract: *void*	param: procObj [IAIProcessorObject* - inout] Allows the AI engine to set the AI observer to be attached to this game object.
aisGetObjectPosition ()	«pure» public abstract: *point3f*	Position data read
aisSetObjectP	«pure» public	param: pos [point3f& - inout]

osition (*point3f&*)	abstract: *void*	Position data write

B - 1.2.2.2.1.2.2.1.4 IAISceneManager

Type: *public abstract «interface»* **Interface**
Package: AI Interfaces The Object System Implements

The scene manager provides the object list for the component to process.

IAISceneManager Interfaces

Method	Type	Notes
aisGetAIProcessableObjects ()	«pure» public abstract: *IAIObjectIterator**	Ask the view's scene manager for a list of objects to process.

B - 1.2.2.2.1.2.2.1.5 IAIView

Type: *public abstract «interface»* **Interface**
Package: AI Interfaces The Object System Implements

The game object system implements this interface to provide "views" into the data. A view is just some context information and acess to a list of objects to process.

IAIView Interfaces

Method	Type	Notes
aisGetAIViewProcessor ()	«pure» public abstract: *IAIViewProcessor **	Get access to the attached domain-specific view observer that will process this view.
aisAssignAIViewProcessor (*IAIViewProcessor**)	«pure» public abstract: *void*	param: viewProc [IAIViewProcessor* - inout] Set the attached domain-specific view observer that will process this view.
aisGetSceneM	«pure» public	Request access to the object list in this view.

anager ()	abstract: *IAIScene Manager* *	

B - 1.2.2.2.2 AI2System

This represents one artifical intelligence logical module. It's functionality will be very simple, possibly adjust the object's orientation in 3-space.

B - 1.2.2.2.1 AI2 Component - Implementation

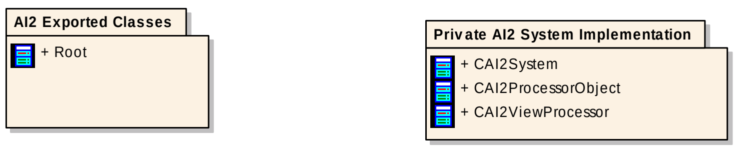


Figure 110 : AI2 Component - Example Implementation

B - 1.2.2.2.2.1.1 AI2 Exported Classes

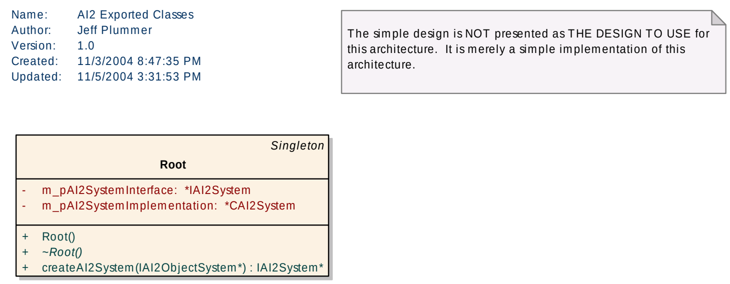


Figure 111 : AI2 Exported Classes

B - 1.2.2.2.2.1.1.1.1 Root

Type: *public* **Class**
Extends: *Singleton.*
Package: AI2 Exported Classes

This class is the only exported class in the Artifical Intelligence component. It represents the initial link to the AI system. From here the game system will connect to the AI system, and request an interface to the AI system. Root is not part of the formal architecture, it is an implementation connection point. In the real world it may be necessary to communicate in more ways with the logical component (due to specific library initializations, etc.). These "extra" communications can be done through the root object directly to the instance of the AI system, rather than through the architectural specified interface.

Root Attributes

Attribute	Type	Notes
m_pAI2SystemInterface	private : *IAI2System*	This is a pointer to the architectural interface to the AI component. The creator of root will receive a pointer to this interface after calling "CreateAISystem".
m_pAI2SystemImplementation	private : *CAI2System*	A pointer to the implementation of the AI system. This should never be accessed publicly. It exists to handle those special

		"real world" occasions where the architectural interface doesn't handle implementation specific features.

Root Methods

Method	Type	Notes
Root ()	public:	Constructor - Create an instance of the AI system.
~Root ()	public abstract:	Destructor - Destroy the instance of the AI System.
createAI2System (*IAI2ObjectSystem**)	public: *IAI2System**	param: pObjectSystem [IAI2ObjectSystem* - inout] Connect the object system to the AI system and return an interface to AI system @param pObjectSystem A pointer to an object that implements the IAIObjectSystem interface. The AI component will use this interface to communicate to the data section of the game system.

B - 1.2.2.2.2.1.2 Private AI2 System Implementation

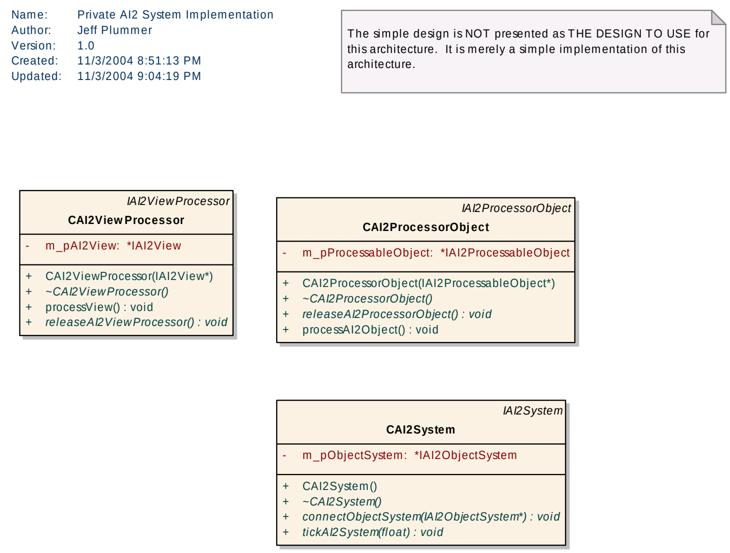


Figure 112 : Private AI2 System Implementation

B - 1.2.2.2.2.1.2.1.1 CAI2System

Type: public **Class**
Extends: *IAI2System.*
Package: Private AI2 System Implementation

This class represents the implementation of the AI system. It implements the IAISystem interface, and will be responsible for performing AI operations on the objects it receives from the Object component.

CAI2System Attributes

Attribute	Type	Notes
m_pObjectSystem	private : *IAI2ObjectSystem*	Pointer to the object system that this AI component is attached to.

CAI2System Methods

Method	Type	Notes
CAI2System	public:	Constrcutor

()		
~CAI2System ()	public abstract:	Destructor
connectObjectSystem (*IAI2ObjectSystem**)	public abstract: *void*	param: objectSystem [IAI2ObjectSystem* - inout] IAISystem interface implementation Connect the AI system to the object component that contains of the AI objects to be processed. @param objectSystem A pointer to an object that implements the IAIObjectSystem interface. The AI component will use this interface to communicate to the data section of the game system.
tickAI2System (*float*)	public abstract: *void*	param: tDiff [float - in] IAISystem interface implementation Causes the AI component to iterate one cycle of time... This will be expanded in the next design iteration of the thesis.

B - 1.2.2.2.2.1.2.1.2 CAI2ProcessorObject

Type: *public* **Class**
Extends: *IAI2ProcessorObject.*
Package: Private AI2 System Implementation

This is the AI Object observer that attaches to a game object. It uses the AI interface into the game object to get access to the necessary data. The AI Processor object will do that AI calculations treating the game object simply as a data access point.

CAI2ProcessorObject Attributes

Attribute	Type	Notes
m_pProcessableObject	private : *IAI2ProcessableObject*	The AI Processable game object this observer is attached to.

CAI2ProcessorObject Methods

Method	**Type**	**Notes**
CAI2ProcessorObject (*IAI2ProcessableObject**)	public:	param: pObject [IAI2ProcessableObject* - inout] Construction/Destruction
~CAI2ProcessorObject ()	public abstract:	Destructor
releaseAI2ProcessorObject ()	public abstract: *void*	The game object should call this function to delete the processor when the game object is deleted.
processAI2Object ()	public: *void*	Perform AI Processing on the game object it is attached to. In this case just rotate the object.

B - 1.2.2.2.2.1.2.1.3 CAI2ViewProcessor

Type: *public* **Class**
Extends: *IAI2ViewProcessor*.
Package: Private AI2 System Implementation

This class attaches to a view and processes the view (i.e. uses the view interface to request objects and works with the object processors attached to the objects).

CAI2ViewProcessor Attributes

Attribute	**Type**	**Notes**
m_pAI2View	private : *IAI2View*	Pointer to the view being observed.

CAI2ViewProcessor Methods

Method	**Type**	**Notes**
CAI2ViewProcessor (*IAI2View**)	public:	param: pView [IAI2View* - inout] Construction/Destruction
~CAI2ViewProcessor ()	public abstract:	Destructor
processView	public:	Perform AI Processing of this

()	*void*	view. Request list of AI capable objects, and call their observer processors.
releaseAI2ViewProcessor ()	public abstract: *void*	Call during view destructor to release this observer.

B - 1.2.2.2.2 AI2 Component - Interfaces

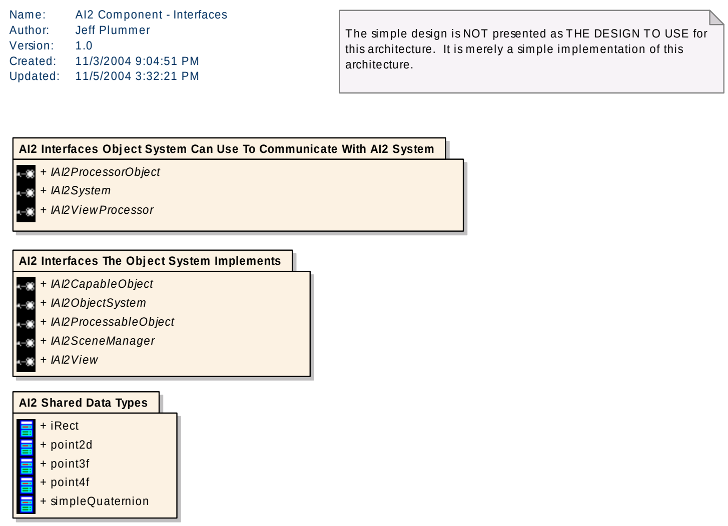


Figure 113 : AI2 Component - Interfaces

B - 1.2.2.2.2.2.1 AI2 Interfaces Object System Can Use To Communicate With AI2 System

This diagram shows the interfaces that are made available to the game system to use in order to communicate with the AI2 System.

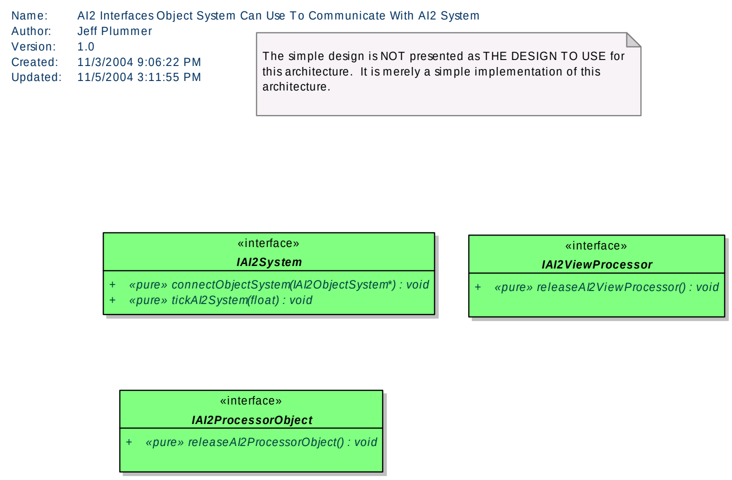


Figure 114 : AI2 Interfaces Object System Can Use To Communicate With AI2 System

B - 1.2.2.2.2.2.1.1.1 IAI2ProcessorObject

Type: *public abstract «interface»* **Interface**
Package: AI2 Interfaces Object System Can Use To Communicate With AI2 System

This is the interface the game system can use to access the domain-specific processor that is attached to a game object. This example is empty, showing that game objects don't necessarily require domain-specific functionality access.

IAI2ProcessorObject Interfaces

Method	Type	Notes
releaseAI2ProcessorObject ()	«pure» public abstract: *void*	Only required in C++ because there is no memory management

B - 1.2.2.2.2.2.1.1.2 IAI2System

Type: *public abstract «interface»* **Interface**
Package: AI2 Interfaces Object System Can Use To Communicate With AI2 System

This interface is the architectural connection from the game system to the AI component. One of the major goals of this architecture is to limit interaction from outside into the AI component. So this interface will provide only the functionality to setup the AI system and provide the AI system with the means to communicate back to the data. From that point on most communication will originate from the AI system back to the data.

IAI2System Interfaces

Method	Type	Notes
connectObject System (*IAI2ObjectSy stem**)	«pure» public abstract: *void*	param: objectSystem [IAI2ObjectSystem* - inout] Use this method to connect an AI Capable Object Management System to the AI Component.
tickAI2System (*float*)	«pure» public abstract: *void*	param: tDiff [float - in] Use this method to Tick the AI2 system, so that it will request and process AI2 objects.

B - 1.2.2.2.2.2.1.1.3 IAI2ViewProcessor

Type: *public abstract «interface»* **Interface**
Package: AI2 Interfaces Object System Can Use To Communicate With AI2 System

This is the interface the game system can use to access the domain-specific view processor that is attached to a view. This example is empty, showing that game views don't necessarily require domain-specific functionality access.

IAI2ViewProcessor Interfaces

Method	Type	Notes
releaseAI2Vie wProcessor ()	«pure» public abstract: *void*	Only required in C++ because there is no memory management. Call this during the game object destructor.

B - 1.2.2.2.2.2.2 AI2 Interfaces The Object System Implements

This diagram shows the interfaces the object system will implement in order to be usable by the AI2 System.

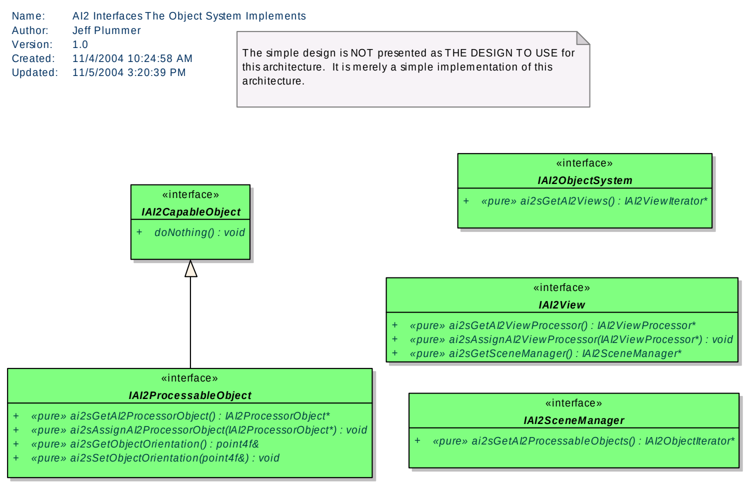


Figure 115 : AI2 Interfaces The Object System Implements

B - 1.2.2.2.2.2.2.1.1 IAI2CapableObject

Type: *public abstract «interface»* **Interface**
Package: AI2 Interfaces The Object System Implements

This class is required for C++ and dynamic type casting. It has no other uses.

IAI2CapableObject Interfaces

Method	Type	Notes
doNothing ()	public abstract: *void*	

B - 1.2.2.2.2.2.2.1.2 IAI2ObjectSystem

Type: *public abstract «interface»* **Interface**
Package: AI2 Interfaces The Object System Implements

This interface is the architectural connection from the object system responsible for managing objects capable of AI to the AI component. Using this interface the AI component will request AI capable objects and perform the appropriate AI operations on them.

IAI2ObjectSystem Interfaces

Method	Type	Notes
ai2sGetAI2Views ()	«pure» public abstract: *IAI2ViewIterator**	Get an iterator (list) of active views to process.

B - 1.2.2.2.2.2.2.1.3 IAI2ProcessableObject

Type: *public abstract «interface»* **Interface**
Extends: *IAI2CapableObject.*
Package: AI2 Interfaces The Object System Implements

Game objects that wish to be processable by this AI engine must implement this interface. It allows the AI system to read/write certain data elements.

IAI2ProcessableObject Interfaces

Method	Type	Notes
ai2sGetAI2ProcessorObject ()	«pure» public abstract: *IAI2ProcessorObject**	Allows the AI engine to get the AI observer object attached to this game object.
ai2sAssignAI2ProcessorObject (*IAI2ProcessorObject**)	«pure» public abstract: *void*	param: procObj [IAI2ProcessorObject* - inout] Allows the AI engine to set the AI observer to be attached to this game object.
ai2sGetObjectOrientation ()	«pure» public abstract: *point4f&*	Orientation data read
ai2sSetObjectOrientation (*point4f&*)	«pure» public abstract: *void*	param: pt [point4f& - inout] Orientation data write

B - 1.2.2.2.2.2.2.1.4 IAI2SceneManager

Type: *public abstract «interface»* **Interface**
Package: AI2 Interfaces The Object System Implements

The scene manager provides the object list for the component to process.

IAI2SceneManager Interfaces

Method	Type	Notes
ai2sGetAI2ProcessableObjects ()	«pure» public abstract: *IAI2ObjectIterator **	Ask the view's scene manager for a list of objects to process.

B - 1.2.2.2.2.2.2.1.5 IAI2View

Type: *public abstract «interface»* **Interface**
Package: AI2 Interfaces The Object System Implements

The game object system implements this interface to provide "views" into the data. A view is just some context information and acess to a list of objects to process.

IAI2View Interfaces

Method	Type	Notes
ai2sGetAI2ViewProcessor ()	«pure» public abstract: *IAI2ViewProcessor**	Get access to the attached domain-specific view observer that will process this view.
ai2sAssignAI2ViewProcessor (*IAI2ViewProcessor**)	«pure» public abstract: *void*	param: viewProc [IAI2ViewProcessor* - inout] Set the attached domain-specific view observer that will process this view.
ai2sGetSceneManager ()	«pure» public abstract:	Request access to the object list in this view.

	*IAI2SceneManager**	

B - 1.2.2.3.2 Game Object System

The Game Object Logical Module will be responsible for managing the game objects. It will provide "views" (or object lists and their contexts) to the various domain-specific modules that are attached.

It could potentially provide object culling etc to make sure each view contains only relavant objects, but for this simple prototype that will not be done.

B - 1.2.2.3.1 Game Object Component - Implementation

B - 1.2.2.2.3.1.1 Game Object Component Exported Classes

Name:	Game Object Component Exported Classes
Author:	Jeff Plummer
Version:	1.0
Created:	11/4/2004 11:08:39 AM
Updated:	11/8/2004 3:41:25 PM

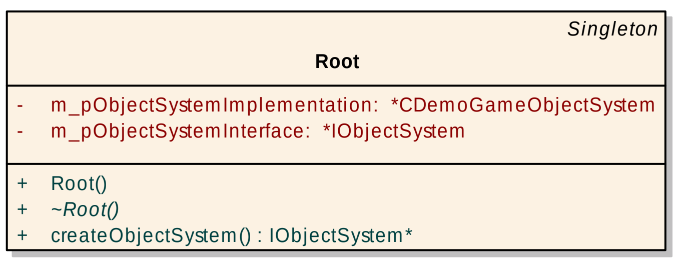


Figure 116 : Game Object Component Exported Classes

B - 1.2.2.2.3.1.1.1.1 <u>Root</u>

Type: *public* **<u>Class</u>**
Extends: *Singleton.*
Package: Game Object Component Exported Classes

This class is the only exported class in the Object component. It represents the initial link to the Object system. From here the game system will connect to the Object system, and request an interface to the Object system. Root is not part of the formal architecture, it is an implementation connection point. In the real world it may be necessary to communicate in more ways with the logical component (due to specific library initializations, etc.). These "extra" communications can be done through the root object directly to the instance of the Object system, rather than through the architectural specified interface.

Root Attributes

Attribute	Type	Notes
m_pObjectSystemImplementation	private : *CDemoGameObjectSystem*	
m_pObjectSystemInterface	private : *IObjectSystem*	

Root Methods

Method	Type	Notes
Root ()	public:	Construction/Destruction
~Root ()	public abstract:	
createObjectSystem ()	public: *IObjectSystem**	

B - 1.2.2.2.3.1.2 Private Game Object Component Implementation

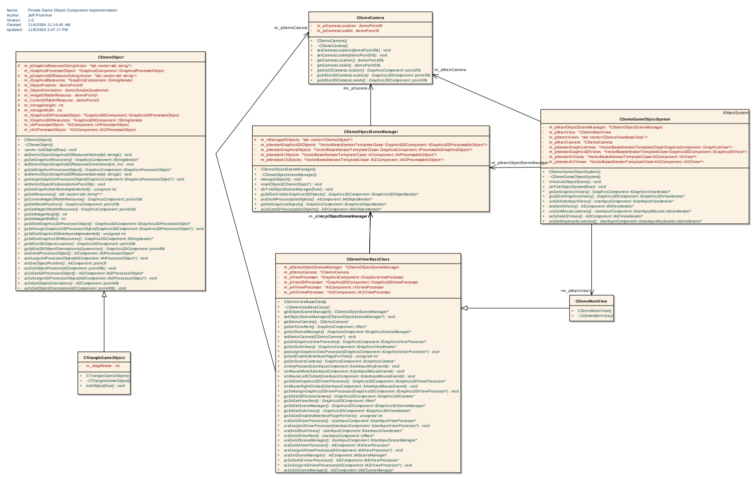


Figure 117 : Private Game Object Component Implementation

B - 1.2.2.2.3.1.2.1.1 CDemoCamera

Type: *public* **Class**
Implements: *I2DGraphicsCamera, IGraphics3DCamera.*
Package: Private Game Object Component Implementation

CDemoCamera Attributes

Attribute	Type	Notes
m_ptCameraLocation	private : *demoPoint3f*	
m_ptCameraLookAt	private : *demoPoint3f*	

CDemoCamera Methods

Method	Type	Notes
CDemoCamera ()	public:	Construction/Destruction

~CDemoCamera ()	public abstract:	
setCameraLocation (*demoPoint3f&*)	public: *void*	param: loc [demoPoint3f& - inout] Setters
setCameraLookAt (*demoPoint3f&*)	public: *void*	param: lookAt [demoPoint3f& - inout]
getCameraLocation ()	public: *demoPoint3f&*	Getters
getCameraLookAt ()	public: *demoPoint3f&*	
gsGet2DCameraLocation ()	public abstract: *GraphicsComponent::point2f&*	Component Interfaces//////////////////// I2DGraphicsCamera
gs3dGet3DCameraLocation ()	public abstract: *Graphics3DComponent::point3f&*	IGraphics3DCamera
gs3dGet3DCameraLookAt ()	public abstract: *Graphics3DComponent::point3f&*	

B - 1.2.2.2.3.1.2.1.2 CDemoGameObjectSystem

Type: *public* **Class**
Extends: *IObjectSystem*. Implements: *IAI2ObjectSystem*, *IAIObjectSystem*,

IGraphics3DObjectSystem, IGraphicsObjectSystem, IObjectSystem, IUserInputObjectSystem.
Package: Private Game Object Component Implementation

CDemoGameObjectSystem Attributes

Attribute	Type	Notes
m_pMainObjectSceneManager	private : *CDemoObjectSceneManager*	
m_pMainView	private : *CDemoMainView*	
m_pDemoViews	private : *std::vector<CDemoViewBaseClass *>*	
m_pMainCamera	private : *CDemoCamera*	
m_pIteratorGraphicsViews	private : *VectorBasedIteratorTemplateClass< GraphicsComponent::IGraphicsView *>*	
m_pIteratorGraphics3DViews	private : *VectorBasedIteratorTemplateClass< Graphics3DComponent::IGraphics3DView**	

	>	
m_pIteratorAIViews	private : *VectorBasedIteratorTemplateClass< AIComponent::IAIView*>*	
m_pIteratorAI2Views	private : *VectorBasedIteratorTemplateClass< AI2Component::IAI2View* >*	

CDemoGameObjectSystem Methods

Method	Type	Notes
CDemoGameObjectSystem ()	public:	Construction/Destruction
~CDemoGameObjectSystem ()	public abstract:	
initializeObjectScene ()	public: *void*	
obTickObjectSystem (*float*)	public abstract: *void*	param: tDiff [float - in]
gsGetGraphicsViews ()	public abstract: *GraphicsComponent::IGraphicsViewIterator**	IGraphicsObjectSystemInterface Overridden Functions //
gs3dGetGraph	public abstract:	IGraphics3DObjectSystemInterface Overridden Functions

icsViews ()	*Graphics3DComponent::IGraphics3DViewIterator**	//
uisGetUserInputViews ()	public abstract: *UserInputComponent::IUserInputViewIterator**	IUserInputObjectSystemInterface Overridden Functions //
aisGetAIViews ()	public abstract: *AIComponent::IAIViewIterator**	IAIObjectSystemInterface Overridden Functions //
uisGetMouseListeners ()	public abstract: *UserInputComponent::IUserInputMouseListenerIterator**	
ai2sGetAI2Views ()	public abstract: *AI2Component::IAI2ViewIterator**	IAI2ObjectSystemInterface Overridden Functions //
uisGetKeyboardListeners ()	public abstract: *UserInputComponent::IUserInputKeyboardListenerIterator**	

B - 1.2.2.2.3.1.2.1.3 CDemoMainView

Type: *public* **Class**
Extends: *CDemoViewBaseClass.*
Package: Private Game Object Component Implementation

CDemoMainView Methods

Method	Type	Notes
CDemoMainView ()	public:	Construction/Destruction
~CDemoMainView ()	public abstract:	

B - 1.2.2.2.3.1.2.1.4 CDemoObject

Type: *public abstract* **Class**
Implements: *I2DGraphicsObject, I2DSpriteGraphicsObject, IAI2ProcessableObject, IAIProcessableObject, IAudioObject, IGraphics3DProcessableObject, IProcessableGraphicsObject.*
Package: Private Game Object Component Implementation

CDemoObject Attributes

Attribute	Type	Notes
m_pGraphicsResourceStringVector	protected : *std::vector<std::string*>*	
m_iGraphicsProcessorObject	private : *GraphicsComponent::IGraphicsProcessorObject*	
	protected	

m_pGraphics3DResourceStringVector	: *std::vector<std::string*>*	
m_iGraphicsResources	private : *GraphicsComponent::IStringIterator*	
m_ObjectPosition	protected : *demoPoint3f*	
m_ObjectOrientation	protected : *demoSimpleQuaternion*	
m_ImageOffsetInResource	protected : *demoPoint2i*	
m_CurrentOffsetInResource	protected : *demoPoint2i*	
m_nImageHeight	protected : *int*	
m_nImageWidth	protected : *int*	
m_iGraphics3DProcessorObject	private : *Graphics3DComponent::IGraphics3DProcessorObject*	
m_iGraphics3DResources	private : *Graphics3DComponent::IStringItera*	

	tor	
m_iAIProcessorObject	private : *AIComponent::IAIProcessorObject*	
m_iAI2ProcessorObject	private : *AI2Component::IAI2ProcessorObject*	

CDemoObject Methods

Method	Type	Notes
CDemoObject ()	public:	Construction/Destruction
~CDemoObject ()	public abstract:	
tickObject (*float*)	«pure» public abstract: *void*	param: tDiff [float - in]
setDemoObjectGraphics2DResourceName (*std::string&*)	public: *void*	param: resName [std::string& - inout]
gsGetGraphicsResources ()	public abstract: *GraphicsComponent::IStringIterator**	IGraphicsObject //
setDemoObjectGraphics2DResourceDimensions (*int*, *int*)	public: *void*	param: w [int - in] param: h [int - in]
gsGetGraphics	public abstract:	

ProcessorObject ()	*GraphicsComponent::IGraphicsProcessorObject**	
setDemoObjectGraphics3DResourceName (*std::string&*)	public: *void*	param: res3DName [std::string& - inout]
gsAssignGraphicsProcessorObject (*GraphicsComponent::IGraphicsProcessorObject**)	public abstract: *void*	param: procObj [GraphicsComponent::IGraphicsProcessorObject* - inout]
setDemoObjectPosition (*demoPoint3f&*)	public: *void*	param: p [demoPoint3f& - inout]
gsGetGraphicInterfacesImplemented ()	public abstract: *unsigned int*	
gsGetResources ()	public abstract: *std::vector<std::string*>**	
gsCurrentImageOffsetInResource ()	public abstract: *GraphicsComponent::point2d&*	I2DSpriteGraphicsObject //
gsGetWorldPosition ()	public abstract: *GraphicsComponent::point2f&*	I2DGraphicsObject //

gsGetImageOffsetInResource ()	public abstract: *GraphicsComponent::point2d&*	
gsGetImageHeight ()	public abstract: *int*	
gsGetImageWidth ()	public abstract: *int*	
gs3dGetGraphics3DProcessorObject ()	public abstract: *Graphics3DComponent::IGraphics3DProcessorObject**	IGraphics3DObject //
gs3dAssignGraphics3DProcessorObject (*Graphics3DComponent::IGraphics3DProcessorObject**)	public abstract: *void*	param: procObj [Graphics3DComponent::IGraphics3DProcessorObject* - inout]
gs3dGetGraphic3DInterfacesImplemented ()	public abstract: *unsigned int*	
gs3dGetGraphics3DResources ()	public abstract: *Graphics3DComponent::IStringIterator**	
gs3dGet3DObjectLocation ()	public abstract: *Graphics3DComp*	

	onent::point3f&	
gs3dGet3DObjectOrientationAsQuaternion ()	public abstract: *Graphics3DComponent::point4f&*	
aisGetAIProcessorObject ()	public abstract: *AIComponent::IAIProcessorObject**	IAIProcessableObject //
aisAssignAIProcessorObject (*AIComponent::IAIProcessorObject**)	public abstract: *void*	param: procObj [AIComponent::IAIProcessorObject* - inout]
aisGetObjectPosition ()	public abstract: *AIComponent::point3f*	
aisSetObjectPosition (*AIComponent::point3f&*)	public abstract: *void*	param: pos [AIComponent::point3f& - inout]
ai2sGetAI2ProcessorObject ()	public abstract: *AI2Component::IAI2ProcessorObject**	IAI2ProcessableObject //
ai2sAssignAI2ProcessorObject (*AI2Component::IAI2ProcessorObject**)	public abstract: *void*	param: procObj [AI2Component::IAI2ProcessorObject* - inout]
	public	

ai2sGetObject Orientation ()	abstract: *AI2Component::point4f&*	
ai2sSetObject Orientation (*AI2Component::point4f&*)	public abstract: *void*	param: pt [AI2Component::point4f& - inout]

B - 1.2.2.2.3.1.2.1.5 CDemoObjectSceneManager

Type: *public* **Class**
Implements: *IAI2SceneManager*, *IAISceneManager*, *IGraphics3DSceneManager*, *IGraphicsSceneManager*.
Package: Private Game Object Component Implementation

CDemoObjectSceneManager Attributes

Attribute	Type	Notes
m_vManaged Objects	protected : *std::vector<CDemoObject *>*	
m_pIteratorGraphics3DObjects	private : *VectorBasedIteratorTemplateClass< Graphics3DComponent::IGraphics3DProcessableObject*>*	
m_pIteratorGraphicsObjects	private : *VectorBasedIteratorTemplateClass<*	

	GraphicsComponent::IProcessableGraphicsObject>*	
m_pIteratorAIObjects	private : *VectorBasedIteratorTemplateClass< AIComponent::IAIProcessableObject*>*	
m_pIteratorAI2Objects	private : *VectorBasedIteratorTemplateClass< AI2Component::IAI2ProcessableObject*>*	

CDemoObjectSceneManager Methods

Method	Type	Notes
CDemoObjectSceneManager ()	public:	Construction/Destruction
~CDemoObjectSceneManager ()	public abstract:	
manageObjects ()	public abstract: *void*	
insertObject (*CDemoObject* *)	public: *void*	param: obj [CDemoObject* - inout]
	public:	param: fdiff [float - in]

obTickObjectSceneManager (*float*)	*void*	
gs3dGetVisibleGraphics3DObjects ()	public abstract: *Graphics3DComponent::IGraphics3DObjectIterator**	
aisGetAIProcessableObjects ()	public abstract: *AIComponent::IAIObjectIterator**	IAISceneManager
gsGetGraphicsObjects ()	public abstract: *GraphicsComponent::IGraphicsObjectIterator**	IGraphicsSceneManager
ai2sGetAI2ProcessableObjects ()	public abstract: *AI2Component::IAI2ObjectIterator**	IAI2SceneManager

B - 1.2.2.2.3.1.2.1.6 CDemoViewBaseClass

Type: *public* **Class**
Implements: *IAI2View*, *IAIView*, *IGraphics3DView*, *IGraphicsView*, *IUserInputView*.
Package: Private Game Object Component Implementation

CDemoViewBaseClass Attributes

Attribute	Type	Notes

m_pDemoObjectSceneManager	private : *CDemoObjectSceneManager*	
m_pDemoCamera	private : *CDemoCamera*	
m_pViewProcessor	private : *GraphicsComponent::IGraphicsViewProcessor*	
m_pView3DProcessor	private : *Graphics3DComponent::IGraphics3DViewProcessor*	
m_pAIViewProcessor	private : *AIComponent::IAIViewProcessor*	
m_pAI2ViewProcessor	private : *AI2Component::IAI2ViewProcessor*	

CDemoViewBaseClass Methods

Method	Type	Notes
CDemoViewBaseClass ()	public:	Construction/Destruction
~CDemoViewBaseClass ()	public abstract:	
getObjectSceneManager ()	public: *CDemoObjectScen*	Gets/Sets

	*eManager**	
setObjectSceneManager (*CDemoObjectSceneManager**)	public: *void*	param: pMgr [CDemoObjectSceneManager* - inout]
getDemoCamera ()	public: *CDemoCamera**	
gsGetViewRect ()	public abstract: *GraphicsComponent::iRect**	GraphicsComponent::IGraphicsView
gsGetSceneManager ()	public abstract: *GraphicsComponent::IGraphicsSceneManager**	
setDemoCamera (*CDemoCamera**)	public: *void*	param: pCamera [CDemoCamera* - inout]
gsGetGraphicsViewProcessor ()	public abstract: *GraphicsComponent::IGraphicsViewProcessor**	Graphics// GraphicsComponent::IGraphicsView
gsGetSubViews ()	public abstract: *GraphicsComponent::IGraphicsViewIterator**	

gsAssignGraphicsViewProcessor (*GraphicsComponent::IGraphicsViewProcessor**)	public abstract: *void*	param: viewProc [GraphicsComponent::IGraphicsViewProcessor* - inout]
gsGetEnabledInterfaceFlagsForView ()	public abstract: *unsigned int*	
gsGetSceneCamera ()	public abstract: *GraphicsComponent::IGraphicsCamera**	
onKeyPressed (*UserInputComponent::IUserInputKeyEvent&*)	public abstract: *void*	param: keyEvent [UserInputComponent::IUserInputKeyEvent& - inout] IUserInput::IUserInputKeyboardListener
onMouseMove (*UserInputComponent::IUserInputMouseEvent&*)	public abstract: *void*	param: event [UserInputComponent::IUserInputMouseEvent& - inout] IUserInput::IUserInputMouseListener
onMouseLeftClicked (*UserInputComponent::IUserInputMouseEvent&*)	public abstract: *void*	param: event [UserInputComponent::IUserInputMouseEvent& - inout]
gs3dGetGraphics3DViewProcessor ()	public abstract: *Graphics3DComponent::IGraphics3DViewP*	Graphics3D// Graphics3DComponent::IGraphics3DView

	*rocessor **	
onMouseRightClicked (*UserInputComponent::IUserInputMouseEvent&*)	public abstract: *void*	param: event [UserInputComponent::IUserInputMouseEvent& - inout]
gs3dAssignGraphics3DViewProcessor (*Graphics3DComponent::IGraphics3DViewProcessor**)	public abstract: *void*	param: viewProc [Graphics3DComponent::IGraphics3DViewProcessor* - inout]
gs3dGet3DSceneCamera ()	public abstract: *Graphics3DComponent::IGraphics3DCamera**	
gs3dGetViewRect ()	public abstract: *Graphics3DComponent::iRect**	
gs3dGetSceneManager ()	public abstract: *Graphics3DComponent::IGraphics3DSceneManager**	
gs3dGetSubViews ()	public abstract: *Graphics3DComponent::I*	

	*Graphics3DViewIterator**	
gs3dGetEnabledInterfaceFlagsForView ()	public abstract: *unsigned int*	
uisGetUIViewProcessor ()	public abstract: *UserInputComponent::IUserInputViewProcessor**	UserInput/// UserInput::IUserInputView
uisAssignUIViewProcessor (*UserInputComponent::IUserInputViewProcessor**)	public abstract: *void*	param: viewProc [UserInputComponent::IUserInputViewProcessor* - inout]
uisGetUISubViews ()	public abstract: *UserInputComponent::IUserInputViewIterator**	
uisGetUIViewRect ()	public abstract: *UserInputComponent::UIRect**	
uisGetUISceneManager ()	public abstract: *UserInputComponent::IUserInputSceneManager**	
	public	AI//

aisGetAIViewProcessor ()	abstract: *AIComponent::IAIViewProcessor**	/ AIComponent::IAIView
aisAssignAIViewProcessor (*AIComponent::IAIViewProcessor**)	public abstract: *void*	param: viewProc [AIComponent::IAIViewProcessor* - inout]
aisGetSceneManager ()	public abstract: *AIComponent::IAISceneManager**	
ai2sGetAI2ViewProcessor ()	public abstract: *AI2Component::IAI2ViewProcessor**	AI2Component::IAI2View
ai2sAssignAI2ViewProcessor (*AI2Component::IAI2ViewProcessor**)	public abstract: *void*	param: viewProc [AI2Component::IAI2ViewProcessor* - inout]
ai2sGetSceneManager ()	public abstract: *AI2Component::IAI2SceneManager**	

B - 1.2.2.2.3.1.2.1.7 CTriangleGameObject

Type: *public* **Class**
Extends: *CDemoObject.* Implements: *I2DGraphicsObject, I2DSpriteGraphicsObject.*
Package: Private Game Object Component Implementation

CTriangleGameObject Attributes

Attribute	Type	Notes
m_idegRotate	private : *int*	

CTriangleGameObject Methods

Method	Type	Notes
CTriangleGameObject ()	public:	Construction/Destruction
~CTriangleGameObject ()	public abstract:	
tickObject (*float*)	public abstract: *void*	param: tDiff [float - in]

B - 1.2.2.2.3.1.2.2 *Data Structures*

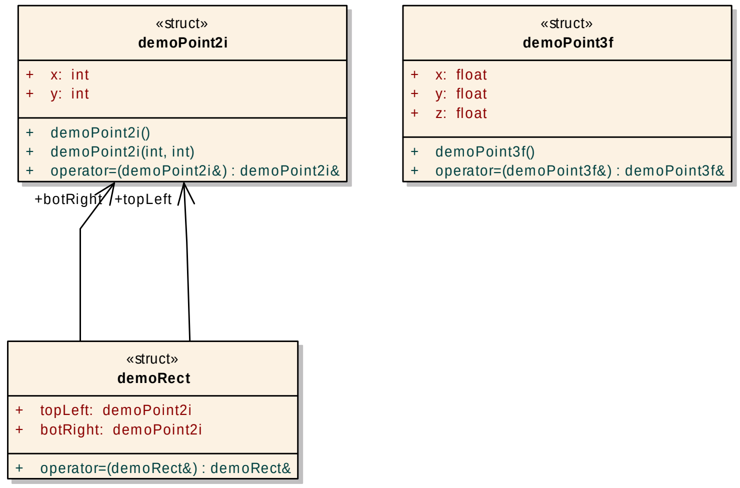


Figure 118 : Game Object System - Data Structures

B - 1.2.2.2.3.1.2.2.1 demoPoint2i

Type: *public «struct»* **Class**
Package: Data Structures

demoPoint2i Attributes

Attribute	Type	Notes
x	public : *int*	
y	public : *int*	

demoPoint2i Methods

Method	Type	Notes
demoPoint2i ()	public:	
demoPoint2i (*int*, *int*)	public:	param: x1 [int - in] param: y1 [int - in]

operator= (*demoPoint2i &*)	public: *demoPoint2i&*	param: pt [demoPoint2i& - inout]

B - 1.2.2.2.3.1.2.2.2 demoPoint3f

Type: *public «struct»* **Class**
Package: Data Structures

demoPoint3f Attributes

Attribute	Type	Notes
x	public : *float*	
y	public : *float*	
z	public : *float*	

demoPoint3f Methods

Method	Type	Notes
demoPoint3f ()	public:	
operator= (*demoPoint3f &*)	public: *demoPoint3f&*	param: pt [demoPoint3f& - inout]

B - 1.2.2.2.3.1.2.2.3 demoRect

Type: *public «struct»* **Class**
Package: Data Structures

demoRect Attributes

Attribute	Type	Notes
topLeft	public : *demoPoint2i*	
botRight	public :	

	demoPoint2i	

demoRect Methods

Method	Type	Notes
operator= (*demoRect&*)	public: *demoRect&*	param: r [demoRect& - inout]

B - 1.2.2.3.2 Game Object Component - Interfaces

Name: Game Object Component - Interfaces
Author: Jeff Plummer
Version: 1.0
Created: 10/19/2004 5:08:08 PM
Updated: 11/8/2004 3:42:16 PM

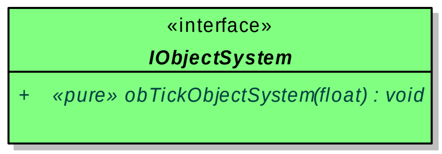


Figure 119 : Game Object Component - Interfaces

B - 1.2.2.2.3.2.1.1.1 IObjectSystem

Type: *public abstract «interface»* **Interface**
Implements: *IGraphicsObjectSystem, IUserInputObjectSystem.*
Package: Game Object Component - Interfaces

IObjectSystem Interfaces

Method	Type	Notes
obTickObjectSystem (*float*)	«pure» public abstract: *void*	param: tDiff [float - in]

B - 1.2.2.3.3 Component Attachings

This class diagram shows how the object system implements the necessary interfaces to interoperate with the AI System.

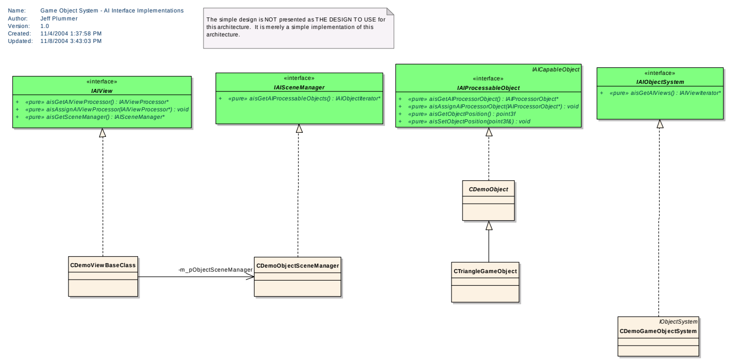


Figure 120 : Game Object System - AI Interface Implementations

This class diagram shows how the object system implements the necessary interfaces to interoperate with the AI2 System.

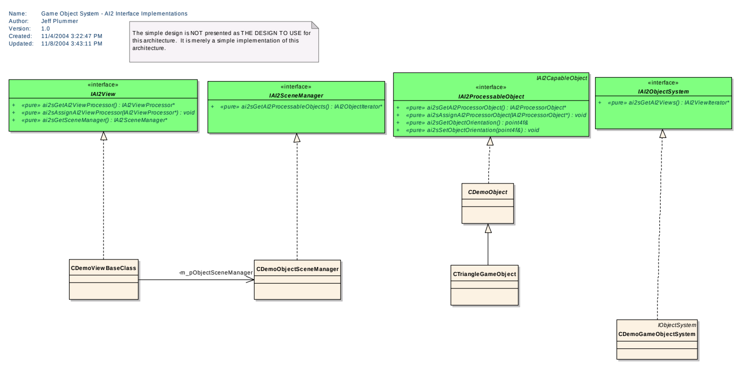


Figure 121 : Game Object System - AI2 Interface Implementations

This class diagram shows how the object system implements the necessary interfaces to interoperate with the 2D graphics System.

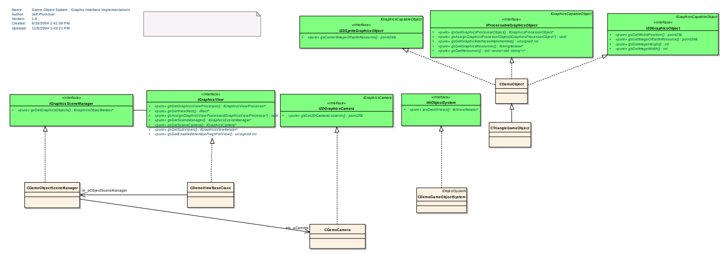


Figure 122 : Game Object System - Graphic Interface Implementations

This class diagram shows how the object system implements the necessary interfaces to interoperate with the Graphics3D System.

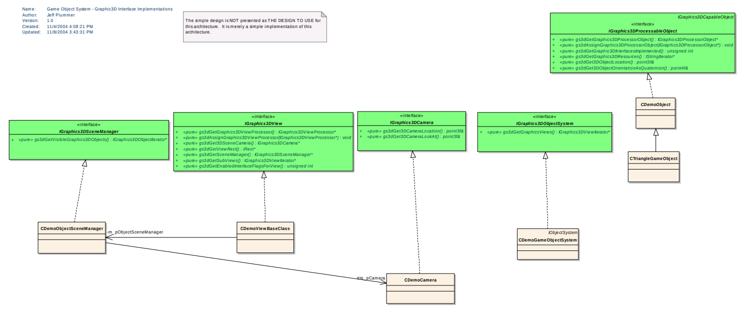


Figure 123 : Game Object System - Graphic3D Interface Implementations

B - 1.2.2.4.2 Game System

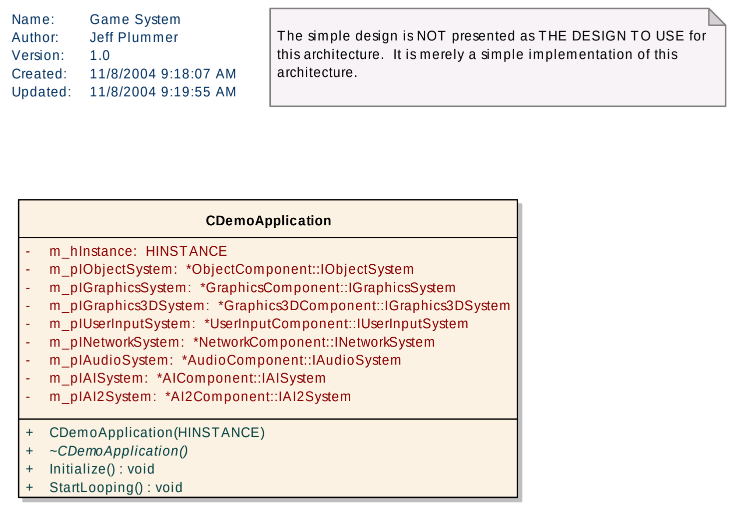


Figure 124 : Game System

B - 1.2.2.2.4.1.1.1.1 <u>CDemoApplication</u>

Type: *public* **Class**
Package: Game System

This class represents the master game system that connects and ticks the various components.

CDemoApplication Attributes

Attribute	Type	Notes
m_hInstance	private : *HINSTANCE*	
m_pIObjectSystem	private : *ObjectComponent::IObjectSystem*	
m_pIGraphicsSystem	private : *GraphicsComponent::IGrap*	

	hicsSystem	
m_pIGraphics3DSystem	private : *Graphics3DComponent::IGraphics3DSystem*	
m_pIUserInputSystem	private : *UserInputComponent::IUserInputSystem*	
m_pINetworkSystem	private : *NetworkComponent::INetworkSystem*	
m_pIAudioSystem	private : *AudioComponent::IAudioSystem*	
m_pIAISystem	private : *AIComponent::IAISystem*	
m_pIAI2System	private : *AI2Component::IAI2System*	

CDemoApplication Methods

Method	Type	Notes
CDemoApplication (*HINSTANCE*)	public:	param: instance [HINSTANCE - in] Construction/Destruction
~CDemoApplication ()	public abstract:	

Initialize ()	public: *void*	Create and connect the necessary components.
StartLooping ()	public: *void*	Tick each component in a loop.

B - 1.2.2.5.2 Graphic 3D System

This represents one graphics 3D logical module. It's functionality will draw objects in 3-Space using an object defined resource.

B - 1.2.2.5.1 Graphics3DComponent - Implementation

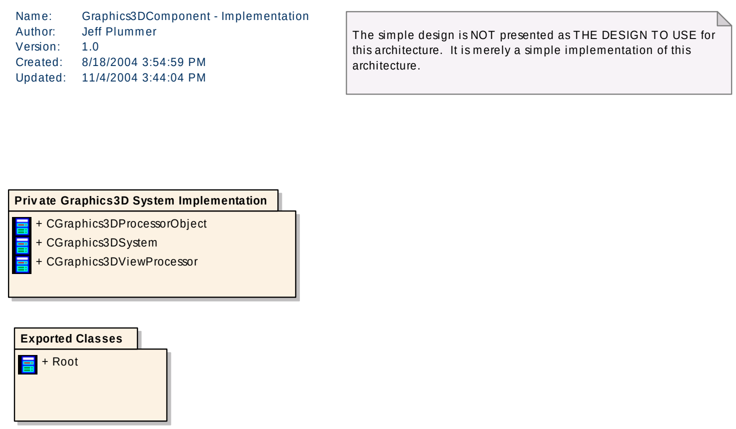


Figure 125 : Graphics3DComponent - Implementation

B - 1.2.2.2.5.1.1 Exported Classes

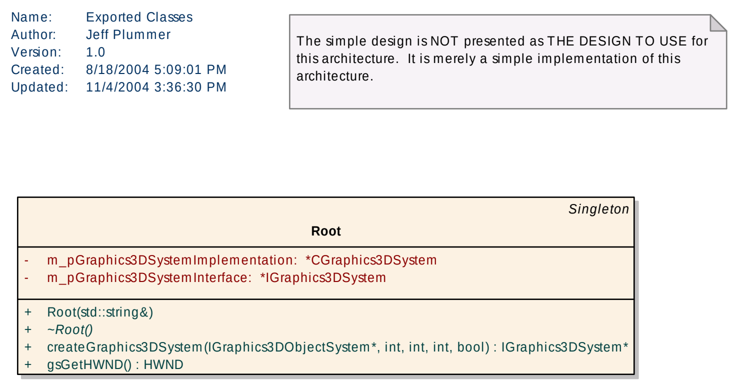


Figure 126 : Exported Classes

B - 1.2.2.2.5.1.1.1.1 Root

Type: *public* **Class**
Extends: *Singleton.*
Package: Exported Classes

This class is the only exported class in the Graphics 3D component. It represents the initial link to the Graphics3D system. From here the game system will connect to the Graphics3D system, and request an interface to the Graphics3D system. Root is not part of the formal architecture, it is an implementation connection point. In the real world it may be necessary to communicate in more ways with the logical component (due to specific library initializations, etc.). These "extra" communications can be done through the root object directly to the instance of the Graphics3D system, rather than through the architectural specified interface.

Root Attributes

Attribute	Type	Notes
m_pGraphics3DSystemImplementation	private : *CGraphics3DSystem*	
m_pGraphics3DSystemInterf	private : *IGraphics3DSyste*	

ace	*m*	

Root Methods

Method	Type	Notes
Root (*std::string&*)	public:	param: resourceConfigFile [std::string& - inout] Construction/Destruction
~Root ()	public abstract:	
createGraphics3DSystem (*IGraphics3DObjectSystem**, *int*, *int*, *int*, *bool*)	public: *IGraphics3DSystem**	param: objectSystem [IGraphics3DObjectSystem* - inout] param: xSize [int - in] param: ySize [int - in] param: bits [int - in] param: fullScreen [bool - in]
gsGetHWND ()	public: *HWND*	

B - 1.2.2.2.5.1.2 Private Graphics3D System Implementation

Figure 127 : Private Graphics3D System Implementation

B - 1.2.2.2.5.1.2.1.1 CGraphics3DProcessorObject

Type: *public* **Class**
Extends: *IGraphics3DProcessorObject.* Implements: *IGraphics3DProcessorObject.*
Package: Private Graphics3D System Implementation

This is the Graphics3D Object observer that attaches to a game object. It uses the Graphics3D interface into the game object to get access to the necessary data. The Graphics3D Processor object will do that calculations treating the game object simply as a data access point.

CGraphics3DProcessorObject Attributes

Attribute	Type	Notes
m_piGraphics3DProcessableObject	private : *IGraphics3DProcessableObject*	
m_pExternalSceneManager	private : *Ogre::CExternalSceneManager*	
m_pEntity	private : *Ogre::Entity*	

m_pExternalSceneNode	private : *Ogre::CExternalSceneManagerNode*	

CGraphics3DProcessorObject Methods

Method	Type	Notes
CGraphics3DProcessorObject (*IGraphics3DProcessableObject*, Ogre::CExternalSceneManager**)	public:	param: obj [IGraphics3DProcessableObject* - inout] param: pSceneManagerConnector [Ogre::CExternalSceneManager* - inout] Construction/Destruction
~CGraphics3DProcessorObject ()	public abstract:	
getExternalSceneNode ()	public: *Ogre::CExternalSceneManagerNode**	
processGraphics3DObject (*IGraphics3DCamera*, unsigned int*)	public: *void*	param: camera [IGraphics3DCamera* - inout] param: ProcessFlags [unsigned int - in]
release3DProcessorObject ()	public abstract: *void*	Only required in C++ because there is no memory management

B - 1.2.2.2.5.1.2.1.2 CGraphics3DSystem

Type: *public* **Class**
Package: Private Graphics3D System Implementation

This class represents the implementation of the Graphics3D system. It implements the IGraphics3DSystem interface, and will be responsible for performing 3D Graphics operations on the objects it receives from the Object component.

CGraphics3DSystem Attributes

Attribute	Type	Notes
m_pGraphics3DObjectSystem	private : *IGraphics3DObjectSystem*	
m_pOgreRoot	private : *Ogre::Root*	
m_pWindow	private : *Ogre::RenderWindow*	
m_pSceneManagerConnector	private : *Ogre::CUseExternalSceneManagers*	
m_viewportMap	private : *VIEWPORTMAP*	
m_availableViewportIDs	private : *std::deque<int>*	
m_usedViewportIDs	private : *std::deque<int>*	

CGraphics3DSystem Methods

Method	Type	Notes
CGraphics3DSystem (*ProgrammingUtilitiesLibrary::String&*)	public:	param: resourceConfigFile [ProgrammingUtilitiesLibrary::String& - inout] Construction/Destruction
	public	

~CGraphics3D System ()	abstract:	
getSceneManagerConnector ()	public: *Ogre::CUseExternalSceneManagers**	
gs3dConnectObject3DSystem (*IGraphics3DObjectSystem**)	public abstract: *void*	param: objectSystem [IGraphics3DObjectSystem* - inout] IGraphics3DSystem
gs3dConfigureAndStartGraphics3DSystem (*int*, *int*, *int*, *bool*)	public abstract: *void*	param: xSize [int - in] param: ySize [int - in] param: bits [int - in] param: fullScreen [bool - in]
gs3dTickGraphics3DSystem (*float*)	public abstract: *void*	param: tDiff [float - in]
processView (*IGraphics3DView**)	private: *void*	param: view [IGraphics3DView* - inout]
setupResources (*ProgrammingUtilitiesLibrary::String&*)	private: *void*	param: resourceConfigFile [ProgrammingUtilitiesLibrary::String& - inout]
configureOgreWindowSettings ()	private: *void*	
getSingletonPtr (*void*)	public static: *CGraphics3DSystem**	param: prm1 [void - in] Singleton Stuff
getSingleton (*void*)	public static:	param: prm1 [void - in]

	CGraphics3DSystem&	

B - 1.2.2.2.5.1.2.1.3 CGraphics3DViewProcessor

Type: *public* **Class**
Extends: *IGraphics3DViewProcessor.* Implements: *IGraphics3DViewProcessor.*
Package: Private Graphics3D System Implementation

This class attaches to a view and processes the view (i.e. uses the view interface to request objects and works with the object processors attached to the objects).

CGraphics3DViewProcessor Attributes

Attribute	Type	Notes
m_piGraphics3DView	private : *IGraphics3DView*	
m_pOgreViewport	private : *Ogre::Viewport*	
m_pOgreCamera	private : *Ogre::Camera*	
m_pOgreExternalSceneManager	private : *Ogre::CExternalSceneManager*	
m_pViewVisibleNodeList	private : *Ogre::ExternalNodeList*	

CGraphics3DViewProcessor Methods

Method	Type	Notes
CGraphics3DViewProcessor (*IGraphics3DView*,*	public:	param: pView [IGraphics3DView* - inout] param: pOgreViewport [Ogre::Viewport* - inout] param: pOgreCamera [

Ogre::Viewport, Ogre::Camera*, Ogre::CExternalSceneManager**)		Ogre::Camera* - inout] param: pOgreExtSceneMgr [Ogre::CExternalSceneManager* - inout] Construction/Destruction
~CGraphics3DViewProcessor ()	public abstract:	
release3DViewProcessor ()	public abstract: *void*	
processView ()	public: *void*	
updateOgreCamera (*IGraphics3DCamera**)	private: *void*	param: cam [IGraphics3DCamera* - inout]

B - 1.2.2.5.2 Graphics3DComponent - Interfaces

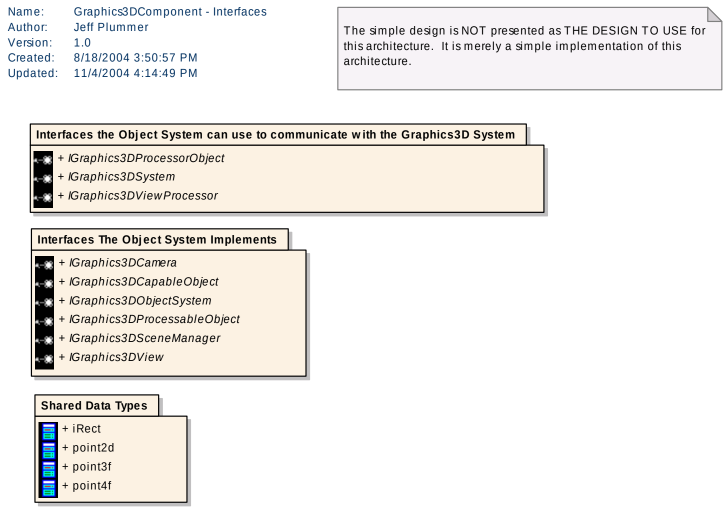


Figure 128 : Graphics3DComponent - Interfaces

B - 1.2.2.2.5.2.1 Interfaces the Object System can use to communicate with the Graphics3D System

This diagram shows the interfaces that are made available to the game system to use in order to communicate with the Graphics3D System.

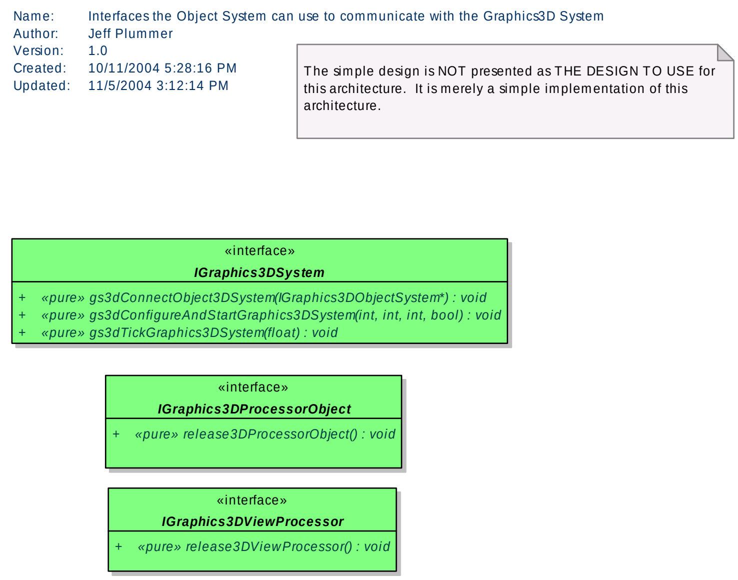


Figure 129 : Interfaces the Object System can use to communicate with the Graphics3D System

B - 1.2.2.2.5.2.1.1.1 <u>IGraphics3DProcessorObject</u>

Type: *public abstract «interface»* **<u>Interface</u>**
Package: Interfaces the Object System can use to communicate with the Graphics3D System

This is the interface the game system can use to access the domain-specific processor that is attached to a game object. This example is empty, showing that game objects don't necessarily require domain-specific functionality access.

IGraphics3DProcessorObject Interfaces

Method	Type	Notes
release3DProcessorObject ()	«pure» public abstract: *void*	Only required in C++ because there is no memory management

B - 1.2.2.2.5.2.1.1.2 IGraphics3DSystem

Type: *public abstract «interface»* **Interface**
Package: Interfaces the Object System can use to communicate with the Graphics3D System

This interface is the architectural connection from the game system to the Graphics3D component. One of the major goals of this architecture is to limit interaction from outside into the Graphics3D component. So this interface will provide only the functionality to setup the Graphics3D system and provide the Graphics3D system with the means to communicate back to the data. From that point on most communication will originate from the Graphics3D system back to the data.

IGraphics3DSystem Interfaces

Method	Type	Notes
gs3dConnectObject3DSystem (*IGraphics3DObjectSystem**)	«pure» public abstract: *void*	param: objectSystem [IGraphics3DObjectSystem* - inout] Use this method to connect an Graphics3D Capable Object Management System to the Graphics3D Component.
gs3dConfigureAndStartGraphics3DSystem (*int*, *int*, *int*, *bool*)	«pure» public abstract: *void*	param: xSize [int - in] param: ySize [int - in] param: bits [int - in] param: fullScreen [bool - in] Configuration of the gaphics engine.
gs3dTickGraphics3DSystem (*float*)	«pure» public abstract: *void*	param: tDiff [float - in] Use this method to Tick the Graphics3D system, so that it will request and process 3D Graphical objects.

B - 1.2.2.2.5.2.1.1.3 IGraphics3DViewProcessor

Type: *public abstract «interface»* **Interface**

Package: Interfaces the Object System can use to communicate with the Graphics3D System

IGraphics3DViewProcessor Interfaces

Method	Type	Notes
release3DViewProcessor ()	«pure» public abstract: *void*	Only required in C++ because there is no memory management

B - 1.2.2.2.5.2.2 Interfaces The Object System Implements

This diagram shows the interfaces the object system will implement in order to be usable by the Graphics3D System.

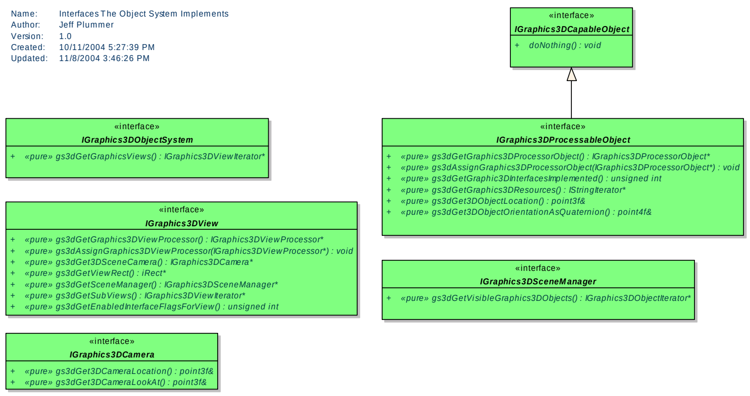


Figure 130 : Interfaces The Object System Implements

B - 1.2.2.2.5.2.2.1.1 <u>IGraphics3DCamera</u>

Type: *public abstract «interface»* **<u>Interface</u>**
Package: Interfaces The Object System Implements

IGraphics3DCamera Interfaces

Method	Type	Notes
gs3dGet3DCameraLocation ()	«pure» public abstract: *point3f&*	
gs3dGet3DCameraLookAt ()	«pure» public abstract: *point3f&*	

B - 1.2.2.2.5.2.2.1.2 <u>IGraphics3DCapableObject</u>

Type: *public abstract «interface»* **<u>Interface</u>**
Package: Interfaces The Object System Implements

IGraphics3DCapableObject Interfaces

Method	Type	Notes
doNothing ()	public abstract: *void*	

B - 1.2.2.2.5.2.2.1.3 IGraphics3DObjectSystem

Type: *public abstract «interface»* **Interface**
Package: Interfaces The Object System Implements

IGraphics3DObjectSystem Interfaces

Method	Type	Notes
gs3dGetGraphicsViews ()	«pure» public abstract: *IGraphics3DViewIterator**	

B - 1.2.2.2.5.2.2.1.4 IGraphics3DProcessableObject

Type: *public abstract «interface»* **Interface**
Extends: *IGraphics3DCapableObject.*
Package: Interfaces The Object System Implements

IGraphics3DProcessableObject Interfaces

Method	Type	Notes
gs3dGetGraphics3DProcessorObject ()	«pure» public abstract: *IGraphics3DProcessorObject**	

gs3dAssignGraphics3DProcessorObject (*IGraphics3DProcessorObject**)	«pure» public abstract: *void*	param: procObj [IGraphics3DProcessorObject* - inout]
gs3dGetGraphic3DInterfacesImplemented ()	«pure» public abstract: *unsigned int*	
gs3dGetGraphics3DResources ()	«pure» public abstract: *IStringIterator**	
gs3dGet3DObjectLocation ()	«pure» public abstract: *point3f&*	
gs3dGet3DObjectOrientationAsQuaternion ()	«pure» public abstract: *point4f&*	

B - 1.2.2.2.5.2.2.1.5 IGraphics3DSceneManager

Type: *public abstract «interface»* **Interface**
Package: Interfaces The Object System Implements

IGraphics3DSceneManager Interfaces

Method	Type	Notes
gs3dGetVisibleGraphics3DObjects ()	«pure» public abstract: *IGraphics3DObjectIterator**	

B - 1.2.2.2.5.2.2.1.6 IGraphics3DView

Type: *public abstract «interface»* **Interface**
Package: Interfaces The Object System Implements

IGraphics3DView Interfaces

Method	Type	Notes
gs3dGetGraphics3DViewProcessor ()	«pure» public abstract: *IGraphics3DViewProcessor**	
gs3dAssignGraphics3DViewProcessor (*IGraphics3DViewProcessor**)	«pure» public abstract: *void*	param: viewProc [IGraphics3DViewProcessor* - inout]
gs3dGet3DSceneCamera ()	«pure» public abstract: *IGraphics3DCamera**	
gs3dGetViewRect ()	«pure» public abstract: *iRect**	
gs3dGetSceneManager ()	«pure» public abstract: *IGraphics3DSceneManager**	
gs3dGetSubViews ()	«pure» public abstract:	

	*IGraphics3DViewIterator**	
gs3dGetEnabledInterfaceFlagsForView ()	«pure» public abstract: *unsigned int*	

B - 1.2.2.6.2 Graphics 2D System

This represents one graphics 2D logical module. It's functionality will draw objects in 2-Space using an object defined resource.

B - 1.2.2.6.1 Graphics Component - Implementation

This package contains an example implementation of the Graphics system. The implementation is not meant to show how to implement an graphics engine, but rather show how a graphics component could be connected using the proposed architecture.

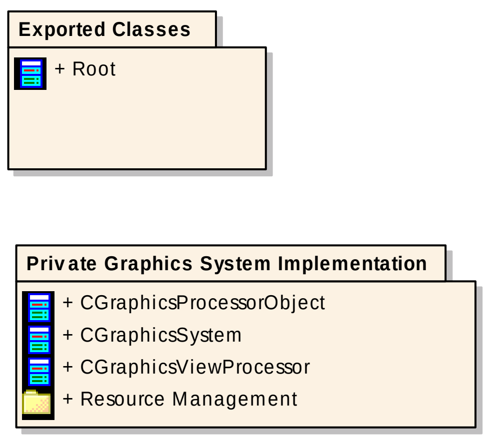


Figure 131 : Graphics Component - Implementation

B - 1.2.2.2.6.1.1 Exported Classes

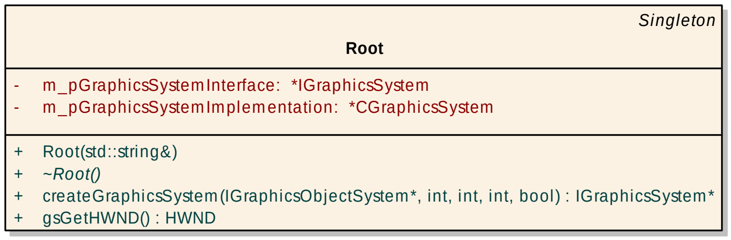


Figure 132 : Exported Classes

B - 1.2.2.2.6.1.1.1.1 Root

Type: *public* **Class**
Extends: *Singleton.*
Package: Exported Classes

This class is the only exported class in the Graphics component. It represents the initial link to the Audio system. From here the game system will connect to the Graphics system, and request an interface to the Graphics system. Root is not part of the formal architecture, it is an implementation connection point. In the real world it may be necessary to communicate in more ways with the logical component (due to specific library initializations, etc.). These "extra" communications can be done through the root object directly to the instance of the Graphics system, rather than through the architectural specified interface.
@author Jeff Plummer
@version 1.0
@updated 11-Feb-2004 07:59:15 PM

Root Attributes

Attribute	Type	Notes
m_pGraphicsSystemInterface	private : *IGraphicsSystem*	
m_pGraphicsSystemImplementation	private : *CGraphicsSystem*	

Root Methods

Method	Type	Notes
Root (*std::string&*)	public:	param: resourceConfigFile [std::string& - inout] Constructor - Create an instance of the Graphics system. @param configFile
~Root ()	public abstract:	Destructor - Destroy the instance of the Graphics System.
createGraphicsSystem (*IGraphicsObjectSystem**, *int*, *int*, *int*, *bool*)	public: *IGraphicsSystem**	param: objectSystem [IGraphicsObjectSystem* - inout] param: xSize [int - in] param: ySize [int - in] param: bits [int - in] param: fullScreen [bool - in] Connect the object system to the Graphics system and return an interface to Graphics system @param objectSystem A pointer to an object that implements the IGraphicsObjectSystem interface. The Graphics component will use this interface to communicate to the data section of the game system. @param xSize The number of pixels in the X direction of the render window. @param ySize The number of pixels in the Y direction of the render window. @param bits The number of bits per pixel data format. @param fullScreen Make the render window full screen or run in a window.
gsGetHWND ()	public: *HWND*	Implementation specific function that returns a handle to the window. Windows(tm) implementation specific.

B - 1.2.2.2.6.1.2 Private Graphics System Implementation

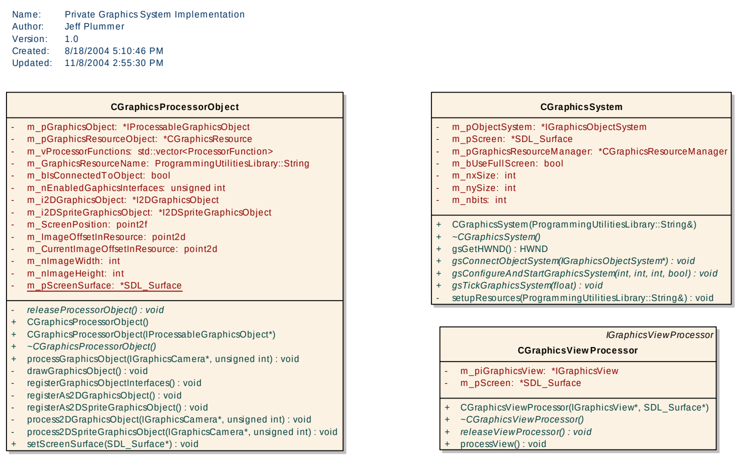


Figure 133 : Private Graphics System Implementation

B - 1.2.2.2.6.1.2.1.1 CGraphicsProcessorObject

Type: *public* **Class**
Implements: *IGraphicsProcessorObject.*
Package: Private Graphics System Implementation

CGraphicsProcessorObject Attributes

Attribute	Type	Notes
m_pGraphicsObject	private : *IProcessableGraphicsObject*	
m_pGraphicsResourceObject	private : *CGraphicsResource*	
m_vProcessorFunctions	private : *std::vector<ProcessorFun*	

	ction>	
m_GraphicsResourceName	private : *ProgrammingUtilitiesLibrary::String*	
m_bIsConnectedToObject	private : *bool*	
m_nEnabledGaphicsInterfaces	private : *unsigned int*	
m_i2DGraphicsObject	private : *I2DGraphicsObject*	
m_i2DSpriteGraphicsObject	private : *I2DSpriteGraphicsObject*	
m_ScreenPosition	private : *point2f*	Variables used for rendering a 2D Image
m_ImageOffsetInResource	private : *point2d*	
m_CurrentImageOffsetInResource	private : *point2d*	
m_nImageWidth	private : *int*	
m_nImageHeight	private : *int*	
m_pScreenSurface	private static : *SDL_Surface*	

CGraphicsProcessorObject Methods

Method	Type	Notes
releaseProcessorObject ()	private abstract: *void*	IGraphicsProcessorObject
CGraphicsProcessorObject ()	public:	Construction/Destruction
CGraphicsProcessorObject (*IProcessableGraphicsObject**)	public:	param: pObject [IProcessableGraphicsObject* - inout]
~CGraphicsProcessorObject ()	public abstract:	
processGraphicsObject (*IGraphicsCamera*, unsigned int*)	public: *void*	param: camera [IGraphicsCamera* - inout] param: ProcessFlags [unsigned int - in]
drawGraphicsObject ()	private: *void*	
registerGraphicsObjectInterfaces ()	private: *void*	
registerAs2DGraphicsObject ()	private: *void*	
registerAs2DSpriteGraphicsObject ()	private: *void*	
process2DGraphicsObject (*IGraphicsCamera*, unsigned int*)	private: *void*	param: camera [IGraphicsCamera* - inout] param: InterfaceEnabledCode [unsigned int - in]

process2DSpriteGraphicsObject (*IGraphicsCamera*, unsigned int*)	private: *void*	param: camera [IGraphicsCamera* - inout] param: InterfaceEnabledCode [unsigned int - in]
setScreenSurface (*SDL_Surface **)	public: *void*	param: screen [SDL_Surface* - inout] Variables used for rendering a 2D sprite

B - 1.2.2.2.6.1.2.1.2 CGraphicsSystem

Type: *public* **Class**
Implements: *IGraphicsSystem.*
Package: Private Graphics System Implementation

This class represents the implementation of the Graphics system. It implements the IGraphicsSystem interface, and allows the game system to communicate with the Graphics component.
@author Jeff Plummer
@version 1.0
@updated 11-Feb-2004 08:33:29 PM

CGraphicsSystem Attributes

Attribute	Type	Notes
m_pObjectSystem	private : *IGraphicsObjectSystem*	Pointer to the interface to the object system. Using this interface the object system will request graphical objects it should draw, etc.
m_pScreen	private : *SDL_Surface*	Implementation Specifc: A pointer to an SDL surface for drawing.
m_pGraphicsResourceManager	private : *CGraphicsResourceManager*	The graphics resource manager object that is responsible for loading and storing in memory the graphics resources.
m_bUseFullSc	private : *bool*	Graphical view is fullscreen or windowed.

reen		
m_nxSize	private : *int*	Screen size in the X direction.
m_nySize	private : *int*	Screen size in the Y direction.
m_nbits	private : *int*	Number of bits per pixel.

CGraphicsSystem Methods

Method	**Type**	**Notes**
CGraphicsSystem (*ProgrammingUtilitiesLibrary::String&*)	public:	param: configFile [ProgrammingUtilitiesLibrary::String& - inout] Constructor @param configFile
~CGraphicsSystem ()	public abstract:	Destructor
gsGetHWND ()	public: *HWND*	Implementation specific*** Returns a handle to the Windows HWND.
gsConnectObjectSystem (*IGraphicsObjectSystem**)	public abstract: *void*	param: objectSystem [IGraphicsObjectSystem* - inout] The architectural interface implementation method that connects the graphics system to the object system. @param objectSystem
gsConfigureAndStartGraphicsSystem (*int*, *int*, *int*, *bool*)	public abstract: *void*	param: xSize [int - in] param: ySize [int - in] param: bits [int - in] param: fullScreen [bool - in] The architectural interface implementation method that configures the graphics system with regard to dimensions and pixel depth. @param xSize @param ySize @param bits

		@param fullScreen
gsTickGraphicsSystem (*float*)	public abstract: *void*	param: tDiff [float - in] The architectural interface implementation method that tells the graphics system to iterate and execute graphics operations on the objects given to it from the object system.
setupResources (*ProgrammingUtilitiesLibrary::String&*)	private: *void*	param: resourceConfigFile [ProgrammingUtilitiesLibrary::String& - inout] Implementation specific*** Method is used to further configure the graphics system. @param configFile

B - 1.2.2.2.6.1.2.1.3 CGraphicsViewProcessor

Type: *public* **Class**
Implements: *IGraphicsViewProcessor*.
Package: Private Graphics System Implementation

CGraphicsViewProcessor Attributes

Attribute	Type	Notes
m_piGraphicsView	private : *IGraphicsView*	
m_pScreen	private : *SDL_Surface*	

CGraphicsViewProcessor Methods

Method	Type	Notes
CGraphicsViewProcessor (*IGraphicsView*,*	public:	param: pView [IGraphicsView* - inout] param: pScreen [SDL_Surface* - inout]

*SDL_Surface**)		Construction/Destruction
~CGraphicsViewProcessor ()	public abstract:	
releaseViewProcessor ()	public abstract: *void*	
processView ()	public: *void*	

B - 1.2.2.6.2 Graphics Component - Interfaces

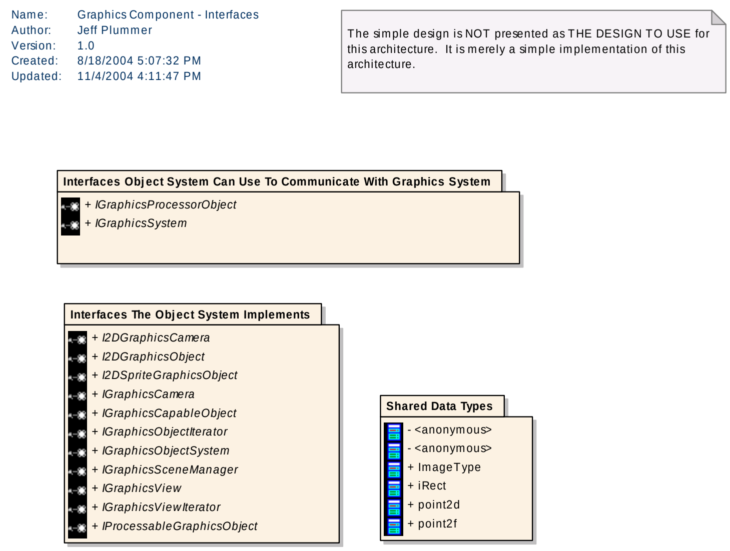


Figure 134 : Graphics Component - Interfaces

B - 1.2.2.2.6.2.1 Interfaces Object System Can Use To Communicate With Graphics System

This diagram shows the interfaces that are made available to the game system to use in order to communicate with the Graphics2D System.

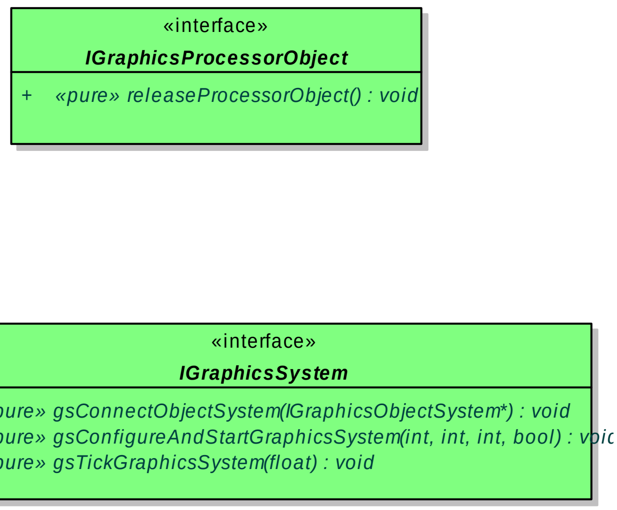


Figure 135 : Interfaces The Graphics System Implements

B - 1.2.2.2.6.2.1.1.1 IGraphicsProcessorObject

Type: *public abstract «interface»* **Interface**
Package: Interfaces Object System Can Use To Communicate With Graphics System

IGraphicsProcessorObject Interfaces

Method	Type	Notes
releaseProcessorObject ()	«pure» public abstract: *void*	Only required in C++ because there is no memory management

B - 1.2.2.2.6.2.1.1.2 IGraphicsSystem

Type: *public abstract «interface»* **Interface**
Package: Interfaces Object System Can Use To Communicate With Graphics System

This interface is the architectural connection from the game system to the Graphics component. One of the major goals of this architecture is to limit interaction from outside into the Graphics component. So this interface will provide only the functionality to setup the Graphics system and provide the Graphics system with the means to communicate back to the data. From that point on most communication will originate from the Graphics system back to the data.
@author Jeff Plummer
@version 1.0
@updated 12-Feb-2004 08:32:46 PM

IGraphicsSystem Interfaces

Method	**Type**	**Notes**
gsConnectObjectSystem (*IGraphicsObjectSystem**)	«pure» public abstract: *void*	param: objectSystem [IGraphicsObjectSystem* - inout] Architectural interface method used to connect the Graphics component to the object system.
gsConfigureAndStartGraphicsSystem (*int*, *int*, *int*, *bool*)	«pure» public abstract: *void*	param: xSize [int - in] param: ySize [int - in] param: bits [int - in] param: fullScreen [bool - in]
gsTickGraphicsSystem (*float*)	«pure» public abstract: *void*	param: tDiff [float - in]

B - 1.2.2.2.6.2.2 Interfaces The Object System Implements

This diagram shows the interfaces the object system will implement in order to be usable by the Graphics2D System.

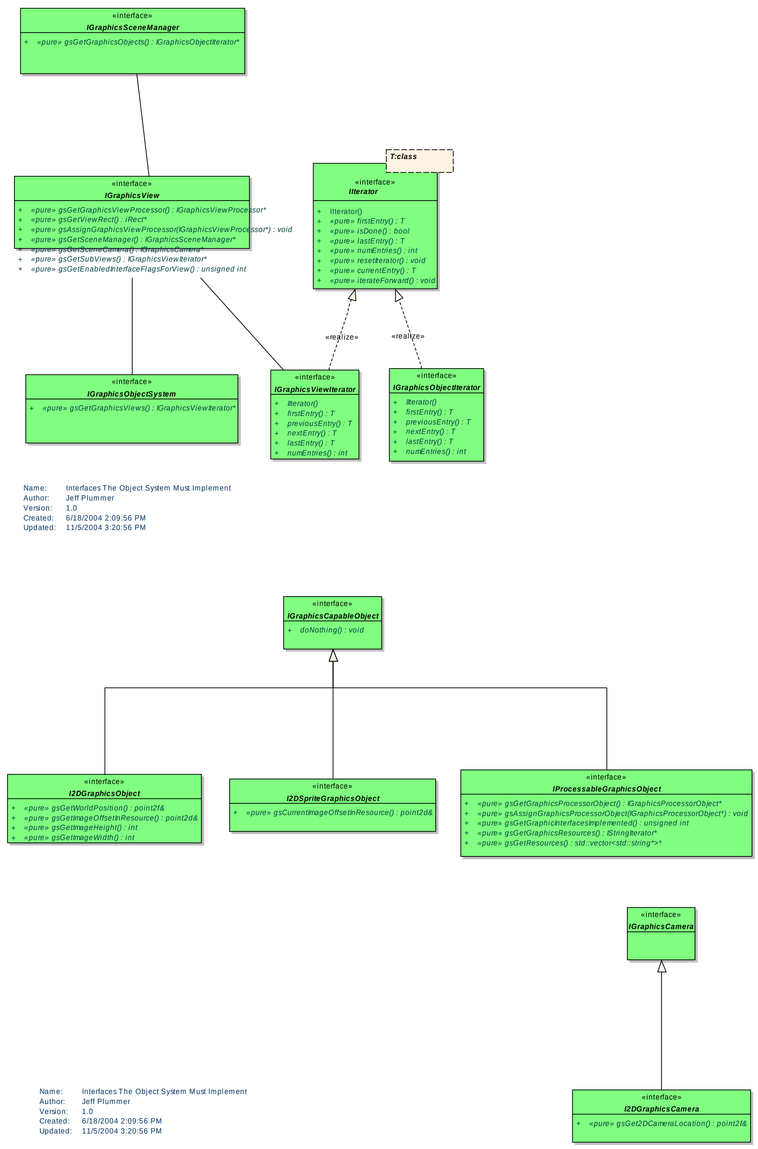


Figure 136 : Interfaces The Object System Must Implement

B - 1.2.2.2.6.2.2.1.1 <u>I2DGraphicsCamera</u>

Type: *public abstract «interface»* **Interface**
Extends: *IGraphicsCamera.*
Package: Interfaces The Object System Implements

I2DGraphicsCamera Interfaces

Method	Type	Notes
gsGet2DCameraLocation ()	«pure» public abstract: *point2f&*	

B - 1.2.2.2.6.2.2.1.2 I2DGraphicsObject

Type: *public abstract «interface»* **Interface**
Extends: *IGraphicsCapableObject.*
Package: Interfaces The Object System Implements

I2DGraphicsObject Interfaces

Method	Type	Notes
gsGetWorldPosition ()	«pure» public abstract: *point2f&*	
gsGetImageOffsetInResource ()	«pure» public abstract: *point2d&*	
gsGetImageHeight ()	«pure» public abstract: *int*	
gsGetImageWidth ()	«pure» public abstract: *int*	

B - 1.2.2.2.6.2.2.1.3 I2DSpriteGraphicsObject

Type: *public abstract «interface»* **Interface**
Extends: *IGraphicsCapableObject.*
Package: Interfaces The Object System Implements

I2DSpriteGraphicsObject Interfaces

Method	Type	Notes
gsCurrentImageOffsetInResource ()	«pure» public abstract: *point2d&*	

B - 1.2.2.2.6.2.2.1.4 IGraphicsCamera

Type: *public abstract «interface»* **Interface**
Package: Interfaces The Object System Implements

B - 1.2.2.2.6.2.2.1.5 IGraphicsCapableObject

Type: *public abstract «interface»* **Interface**
Package: Interfaces The Object System Implements

IGraphicsCapableObject Interfaces

Method	Type	Notes
doNothing ()	public abstract: *void*	

B - 1.2.2.2.6.2.2.1.6 IGraphicsObjectIterator

Type: *public abstract «interface»* **Interface**
Implements: *IIterator.*
Package: Interfaces The Object System Implements

IGraphicsObjectIterator Interfaces

Method	Type	Notes
IIterator ()	public:	
firstEntry ()	public abstract: *T*	
previousEntry ()	public abstract: *T*	
nextEntry ()	public abstract: *T*	
lastEntry ()	public abstract: *T*	
numEntries ()	public abstract: *int*	

B - 1.2.2.2.6.2.2.1.7 IGraphicsObjectSystem

Type: *public abstract «interface»* **Interface**
Package: Interfaces The Object System Implements

This interface is the architectural connection from the object system responsible for managing objects capable of Graphics to the Graphics component. Using this interface the Graphics component will request Graphics capable objects and perform the appropriate Graphics operations on them.
@author Jeff Plummer
@version 1.0
@updated 05-Mar-2004 09:31:42 PM

IGraphicsObjectSystem Interfaces

Method	Type	Notes
gsGetGraphicsViews ()	«pure» public abstract: *IGraphicsViewIterator**	

B - 1.2.2.2.6.2.2.1.8 IGraphicsSceneManager

Type: *public abstract «interface»* **Interface**
Package: Interfaces The Object System Implements

IGraphicsSceneManager Interfaces

Method	Type	Notes
gsGetGraphicsObjects ()	«pure» public abstract: *IGraphicsObjectIterator**	

B - 1.2.2.2.6.2.2.1.9 IGraphicsView

Type: *public abstract «interface»* **Interface**
Package: Interfaces The Object System Implements

IGraphicsView Interfaces

Method	Type	Notes
gsGetGraphicsViewProcessor ()	«pure» public abstract: *IGraphicsViewProcessor**	
gsGetViewRect ()	«pure» public abstract: *iRect**	
gsAssignGraphicsViewProcessor (*IGraphicsViewProcessor**)	«pure» public abstract: *void*	param: viewProc [IGraphicsViewProcessor* - inout]
gsGetSceneManager ()	«pure» public abstract: *IGraphic*	

	*sSceneManager**	
gsGetSceneCamera ()	«pure» public abstract: *IGraphicsCamera **	
gsGetSubViews ()	«pure» public abstract: *IGraphicsViewIterator**	
gsGetEnabledInterfaceFlagsForView ()	«pure» public abstract: *unsigned int*	

B - 1.2.2.2.6.2.2.1.10 IGraphicsViewIterator

Type: *public abstract «interface»* **Interface**
Implements: *IIterator*.
Package: Interfaces The Object System Implements

IGraphicsViewIterator Interfaces

Method	Type	Notes
IIterator ()	public:	
firstEntry ()	public abstract: *T*	
previousEntry ()	public abstract: *T*	
nextEntry ()	public abstract: *T*	
lastEntry ()	public abstract: *T*	

numEntries ()	public abstract: *int*	

B - 1.2.2.2.6.2.2.1.11 IProcessableGraphicsObject

Type: *public abstract «interface»* **Interface**
Extends: *IGraphicsCapableObject.*
Package: Interfaces The Object System Implements

IProcessableGraphicsObject Interfaces

Method	Type	Notes
gsGetGraphicsProcessorObject ()	«pure» public abstract: *IGraphicsProcessorObject**	
gsAssignGraphicsProcessorObject (*IGraphicsProcessorObject**)	«pure» public abstract: *void*	param: procObj [IGraphicsProcessorObject* - inout]
gsGetGraphicInterfacesImplemented ()	«pure» public abstract: *unsigned int*	
gsGetGraphicsResources ()	«pure» public abstract: *IStringIterator**	
gsGetResources ()	«pure» public abstract: *std::vector<std::string*>**	

B - 1.2.2.3 Utility Includes

This package represents a few template classes or generic classes that were shared accross projects.

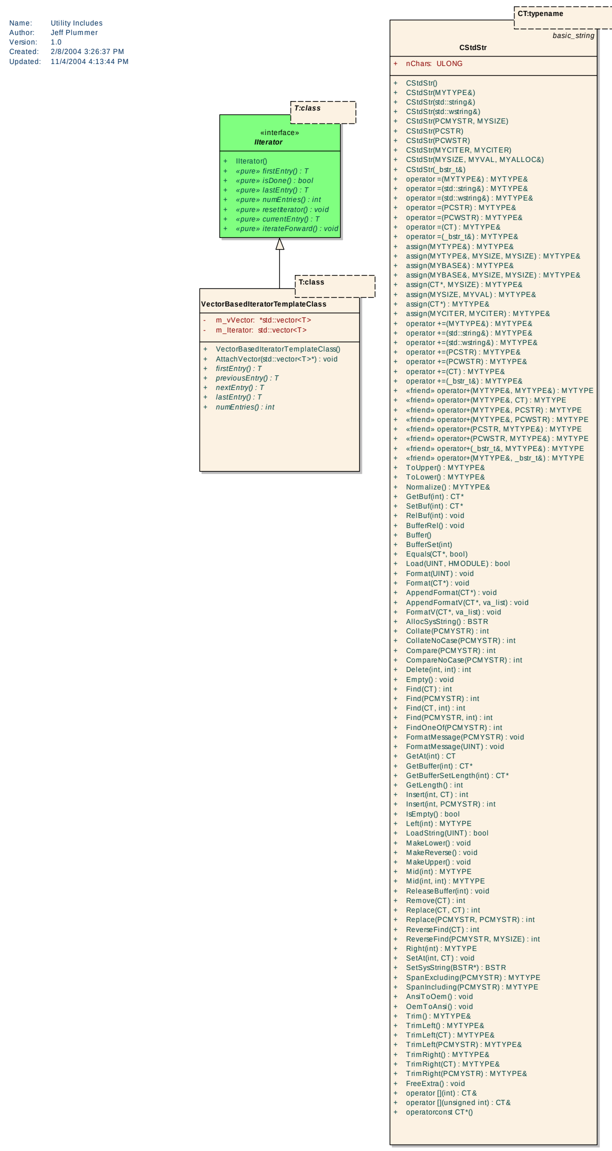


Figure 137 : Utility Includes

B - 1.2.2.3.1.1.1.1.1 <u>CStdStr</u>

Type: *public* **Class**
Implements: *basic_string.*
Package: Utility Includes

#define CStdStr _SS // avoid compiler warning 4786

CStdStr Attributes

Attribute	Type	Notes
nChars	public : *ULONG*	struct SSSHDR - useful for non Std C++ persistence schemes.

CStdStr Methods

Method	Type	Notes
CStdStr ()	public:	CStdStr inline constructors
CStdStr (*MYTYPE&*)	public:	param: str [MYTYPE& - inout]
CStdStr (*std::string&*)	public:	param: str [std::string& - inout]
CStdStr (*std::wstring&*)	public:	param: str [std::wstring& - inout]
CStdStr (*PCMYSTR, MYSIZE*)	public:	param: pT [PCMYSTR - in] param: n [MYSIZE - in]
CStdStr (*PCSTR*)	public:	param: pA [PCSTR - in]
CStdStr (*PCWSTR*)	public:	param: pW [PCWSTR - in]
CStdStr (*MYCITER, MYCITER*)	public:	param: first [MYCITER - in] param: last [MYCITER - in]
CStdStr (*MYSIZE, MYVAL, MYALLOC&*)	public:	param: nSize [MYSIZE - in] param: ch [MYVAL - in] param: al [MYALLOC& - inout]
CStdStr (*_bstr_t&*)	public:	param: bstr [_bstr_t& - inout]
operator = (*MYTYPE&*)	public: *MYTYPE &*	param: str [MYTYPE& - inout] CStdStr inline assignment operators -- the ssasn function now takes care of fixing the

		MSVC assignment bug (see knowledge base article Q172398).
operator = (*std::string&*)	public: *MYTYPE &*	param: str [std::string& - inout]
operator = (*std::wstring&*)	public: *MYTYPE &*	param: str [std::wstring& - inout]
operator = (*PCSTR*)	public: *MYTYPE &*	param: pA [PCSTR - in]
operator = (*PCWSTR*)	public: *MYTYPE &*	param: pW [PCWSTR - in]
operator = (*CT*)	public: *MYTYPE &*	param: t [CT - in]
operator = (*_bstr_t&*)	public: *MYTYPE &*	param: bstr [_bstr_t& - inout]
assign (*MYTYPE&*)	public: *MYTYPE &*	param: str [MYTYPE& - inout] Overloads also needed to fix the MSVC assignment bug (KB: Q172398) Thanks to Pete The Plumber for catching this one *** They also are compiled if you have explicitly turned off refcounting
assign (*MYTYPE&, MYSIZE, MYSIZE*)	public: *MYTYPE &*	param: str [MYTYPE& - inout] param: nStart [MYSIZE - in] param: nChars [MYSIZE - in]
assign (*MYBASE&*)	public: *MYTYPE &*	param: str [MYBASE& - inout]
assign (*MYBASE&, MYSIZE, MYSIZE*)	public: *MYTYPE &*	param: str [MYBASE& - inout] param: nStart [MYSIZE - in] param: nChars [MYSIZE - in]
assign (*CT*, MYSIZE*)	public: *MYTYPE &*	param: pC [CT* - inout] param: nChars [MYSIZE - in]

assign (*MYSIZE, MYVAL*)	public: *MYTYPE &*	param: nChars [MYSIZE - in] param: val [MYVAL - in]
assign (*CT**)	public: *MYTYPE &*	param: pT [CT* - inout]
assign (*MYCITER, MYCITER*)	public: *MYTYPE &*	param: iterFirst [MYCITER - in] param: iterLast [MYCITER - in]
operator += (*MYTYPE&*)	public: *MYTYPE &*	param: str [MYTYPE& - inout] --- CStdStr inline concatenation. --
operator += (*std::string&*)	public: *MYTYPE &*	param: str [std::string& - inout]
operator += (*std::wstring&*)	public: *MYTYPE &*	param: str [std::wstring& - inout]
operator += (*PCSTR*)	public: *MYTYPE &*	param: pA [PCSTR - in]
operator += (*PCWSTR*)	public: *MYTYPE &*	param: pW [PCWSTR - in]
operator += (*CT*)	public: *MYTYPE &*	param: t [CT - in]
operator += (*_bstr_t&*)	public: *MYTYPE &*	param: bstr [_bstr_t& - inout]
operator+ (*MYTYPE&, MYTYPE&*)	«friend» public: *MYTYPE*	param: str1 [MYTYPE& - inout] param: str2 [MYTYPE& - inout] addition operators -- global friend functions.
operator+	«friend»	param: str [MYTYPE& - inout]

(*MYTYPE&*, *CT*)	public: *MYTYPE*	param: t [CT - in]
operator+ (*MYTYPE&*, *PCSTR*)	«friend» public: *MYTYPE*	param: str [MYTYPE& - inout] param: sz [PCSTR - in]
operator+ (*MYTYPE&*, *PCWSTR*)	«friend» public: *MYTYPE*	param: str [MYTYPE& - inout] param: sz [PCWSTR - in]
operator+ (*PCSTR*, *MYTYPE&*)	«friend» public: *MYTYPE*	param: pA [PCSTR - in] param: str [MYTYPE& - inout]
operator+ (*PCWSTR*, *MYTYPE&*)	«friend» public: *MYTYPE*	param: pW [PCWSTR - in] param: str [MYTYPE& - inout]
operator+ (*_bstr_t&*, *MYTYPE&*)	«friend» public: *MYTYPE*	param: bstr [_bstr_t& - inout] param: str [MYTYPE& - inout]
operator+ (*MYTYPE&*, *_bstr_t&*)	«friend» public: *MYTYPE*	param: str [MYTYPE& - inout] param: bstr [_bstr_t& - inout]
ToUpper ()	public: *MYTYPE &*	-- Case changing functions -- ---
ToLower ()	public: *MYTYPE &*	
Normalize ()	public: *MYTYPE &*	
GetBuf (*int*)	public: *CT**	param: nMinLen [int - in] -- CStdStr -- Direct access to character buffer. In the MS' implementation, the at() function that we use here also calls _Freeze() providing us some protection from multithreading problems associated with ref-counting. ----------------------------

		-- -----
SetBuf (*int*)	public: *CT**	param: nLen [int - in]
RelBuf (*int*)	public: *void*	param: nNewLen [int - in]
BufferRel ()	public: *void*	
Buffer ()	public:	
BufferSet (*int*)	public:	param: nLen [int - in]
Equals (*CT*, *bool*)	public query:	param: pT [CT* - inout] param: bUseCase [bool - in]
Load (*UINT*, *HMODULE*)	public: *bool*	param: nId [UINT - in] param: hModule [HMODULE - in] -- --------------------------------- FUNCTION: CStdStr::Load REMARKS: Loads string from resource specified by nID PARAMETERS: nID - resource Identifier. Purely a Win32 thing in this case RETURN VALUE: true if successful, false otherwise -- ---------------------------------
Format (*UINT*)	public: *void*	param: nId [UINT - in] -- --------------------------------- FUNCTION: CStdStr::Format void _cdecl Formst(CStdStringA& PCSTR szFormat, ...) void _cdecl Format(PCSTR szFormat); DESCRIPTION: This function does sprintf/wsprintf style formatting on CStdStringA objects. It looks a lot like MFC's CString::Format. Some people might even call this identical. Fortunately, these people are

		now dead. PARAMETERS: nId - ID of string resource holding the format string szFormat - a PCSTR holding the format specifiers argList - a va_list holding the arguments for the format specifiers. RETURN VALUE: None. -- formatting (using wsprintf style formatting)
Format (*CT**)	public: *void*	param: szFmt [CT* - inout]
AppendFormat (*CT**)	public: *void*	param: szFmt [CT* - inout]
AppendFormatV (*CT**, *va_list*)	public: *void*	param: szFmt [CT* - inout] param: argList [va_list - in] an efficient way to add formatted characters to the string. You may only add up to STD_BUF_SIZE characters at a time, though
FormatV (*CT**, *va_list*)	public: *void*	param: szFormat [CT* - inout] param: argList [va_list - in] -- FUNCTION: FormatV void FormatV(PCSTR szFormat, va_list, argList); DESCRIPTION: This function formats the string with sprintf style format-specs. It makes a general guess at required buffer size and then tries successively larger buffers until it finds one big enough or a threshold (MAX_FMT_TRIES) is exceeded. PARAMETERS: szFormat - a PCSTR holding the format of the output argList - a Microsoft specific va_list for variable argument lists

		RETURN VALUE: --
AllocSysString ()	public query: *BSTR*	-- CString Facade Functions: The following methods are intended to allow you to use this class as a drop-in replacement for CString. --
Collate (*PCMYSTR*)	public query: *int*	param: szThat [PCMYSTR - in]
CollateNoCase (*PCMYSTR*)	public query: *int*	param: szThat [PCMYSTR - in]
Compare (*PCMYSTR*)	public query: *int*	param: szThat [PCMYSTR - in]
CompareNoCase (*PCMYSTR*)	public query: *int*	param: szThat [PCMYSTR - in]
Delete (*int*, *int*)	public: *int*	param: nIdx [int - in] param: nCount [int - in]
Empty ()	public: *void*	
Find (*CT*)	public query: *int*	param: ch [CT - in]
Find (*PCMYSTR*)	public query: *int*	param: szSub [PCMYSTR - in]
Find (*CT*, *int*)	public query: *int*	param: ch [CT - in] param: nStart [int - in]
Find (*PCMYSTR*, *int*)	public query: *int*	param: szSub [PCMYSTR - in] param: nStart [int - in]
FindOneOf (*PCMYSTR*)	public query: *int*	param: szCharSet [PCMYSTR - in]
FormatMessage (*PCMYSTR*)	public: *void*	param: szFormat [PCMYSTR - in]

FormatMessage (*UINT*)	public: *void*	param: nFormatId [UINT - in]
GetAt (*int*)	public query: *CT*	param: nIdx [int - in] --- GetXXXX -- Direct access to character buffer --- -------------
GetBuffer (*int*)	public: *CT**	param: nMinLen [int - in]
GetBufferSetLength (*int*)	public: *CT**	param: nLen [int - in]
GetLength ()	public query: *int*	GetLength() -- MFC docs say this is the # of BYTES but in truth it is the number of CHARACTERs (chars or wchar_ts)
Insert (*int*, *CT*)	public: *int*	param: nIdx [int - in] param: ch [CT - in]
Insert (*int*, *PCMYSTR*)	public: *int*	param: nIdx [int - in] param: sz [PCMYSTR - in]
IsEmpty ()	public query: *bool*	
Left (*int*)	public query: *MYTYPE*	param: nCount [int - in]
LoadString (*UINT*)	public: *bool*	param: nId [UINT - in]
MakeLower ()	public: *void*	
MakeReverse ()	public: *void*	
MakeUpper ()	public: *void*	
Mid (*int*)	public query: *MYTYPE*	param: nFirst [int - in]

Mid (*int*, *int*)	public query: *MYTYPE*	param: nFirst [int - in] param: nCount [int - in]
ReleaseBuffer (*int*)	public: *void*	param: nNewLen [int - in]
Remove (*CT*)	public: *int*	param: ch [CT - in]
Replace (*CT*, *CT*)	public: *int*	param: chOld [CT - in] param: chNew [CT - in]
Replace (*PCMYSTR*, *PCMYSTR*)	public: *int*	param: szOld [PCMYSTR - in] param: szNew [PCMYSTR - in]
ReverseFind (*CT*)	public query: *int*	param: ch [CT - in]
ReverseFind (*PCMYSTR*, *MYSIZE*)	public query: *int*	param: szFind [PCMYSTR - in] param: pos [MYSIZE - in] ReverseFind overload that's not in CString but might be useful
Right (*int*)	public query: *MYTYPE*	param: nCount [int - in]
SetAt (*int*, *CT*)	public: *void*	param: nIndex [int - in] param: ch [CT - in]
SetSysString (*BSTR**)	public query: *BSTR*	param: pbstr [BSTR* - inout]
SpanExcluding (*PCMYSTR*)	public query: *MYTYPE*	param: szCharSet [PCMYSTR - in]
SpanIncluding (*PCMYSTR*)	public query: *MYTYPE*	param: szCharSet [PCMYSTR - in]
AnsiToOem ()	public: *void*	CString's OemToAnsi and AnsiToOem functions are available only in Unicode builds. However since we're a template we also need a runtime check of CT and a reinterpret_cast to account for the fact that

		CStdStringW gets instantiated even in non-Unicode builds.
OemToAnsi ()	public: *void*	
Trim ()	public: *MYTYPE &*	--- Trim and its variants ---
TrimLeft ()	public: *MYTYPE &*	
TrimLeft (*CT*)	public: *MYTYPE &*	param: tTrim [CT - in]
TrimLeft (*PCMYSTR*)	public: *MYTYPE &*	param: szTrimChars [PCMYSTR - in]
TrimRight ()	public: *MYTYPE &*	
TrimRight (*CT*)	public: *MYTYPE &*	param: tTrim [CT - in]
TrimRight (*PCMYSTR*)	public: *MYTYPE &*	param: szTrimChars [PCMYSTR - in]
FreeExtra ()	public: *void*	
operator [] (*int*)	public: *CT&*	param: nIdx [int - in] Array-indexing operators. Required because we defined an implicit cast to operator const CT* (Thanks to Julian Selman for pointing this out)
operator [] (*int*)	public const query: *CT&*	param: nIdx [int - in]
operator [] (*unsigned int*)	public: *CT&*	param: nIdx [unsigned int - in]
operator [] (*unsigned int*)	public const	param: nIdx [unsigned int - in]

	query: *CT&*	
operatorconst CT* ()	public query:	

B - 1.2.2.3.1.1.1.1.2 IIterator

Type: *public abstract «interface»* **Class**
Package: Utility Includes

IIterator Methods

Method	Type	Notes
IIterator ()	public:	
firstEntry ()	«pure» public abstract: *T*	
isDone ()	«pure» public abstract: *bool*	
lastEntry ()	«pure» public abstract: *T*	virtual T previousEntry() = 0; virtual T nextEntry() = 0;
numEntries ()	«pure» public abstract: *int*	virtual bool hasNextEntry() = 0; virtual bool hasPreviousEntry() = 0;
resetIterator ()	«pure» public abstract: *void*	
currentEntry ()	«pure» public abstract: *T*	
iterateForward ()	«pure» public abstract: *void*	

B - 1.2.2.3.1.1.1.1.3 VectorBasedIteratorTemplateClass

Type: *public* **Class**
Extends: *IIterator*.
Package: Utility Includes

VectorBasedIteratorTemplateClass Attributes

Attribute	Type	Notes
m_vVector	private : *std::vector<T>*	
m_Iterator	private : *std::vector<T>*	

VectorBasedIteratorTemplateClass Methods

Method	Type	Notes
VectorBasedIteratorTemplateClass ()	public:	
AttachVector (*std::vector<T>**)	public: *void*	param: v [std::vector<T>* - inout]
firstEntry ()	public abstract: *T*	
previousEntry ()	public abstract: *T*	
nextEntry ()	public abstract: *T*	
lastEntry ()	public abstract: *T*	
numEntries ()	public abstract: *int*	

B - 1.2.3 Dynamic View

B - 1.2.3.1 Initialize

This package contains all use cases that are related to initailizing the game system.

This diagram shows the use cases involved in initializing the prototype.

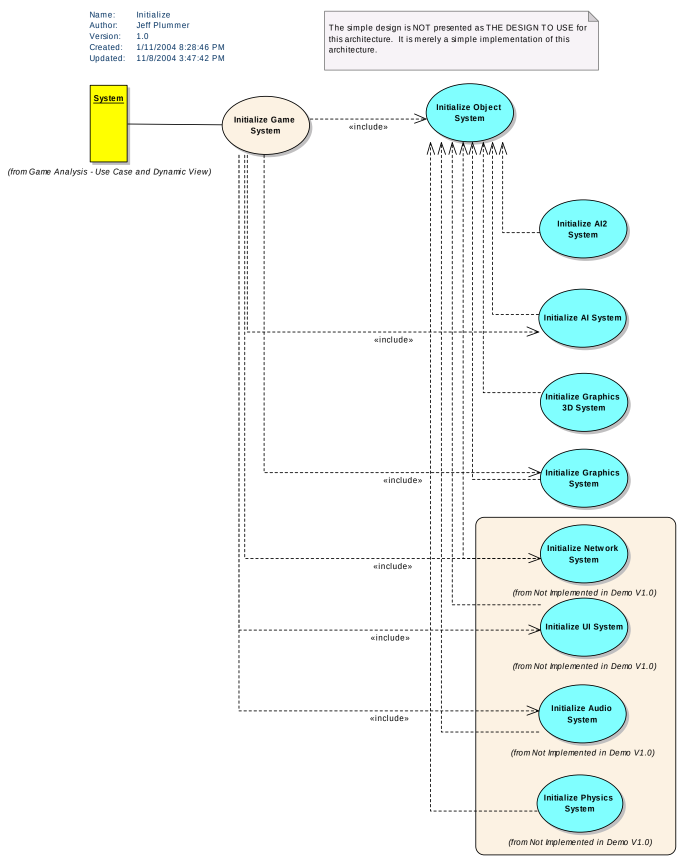
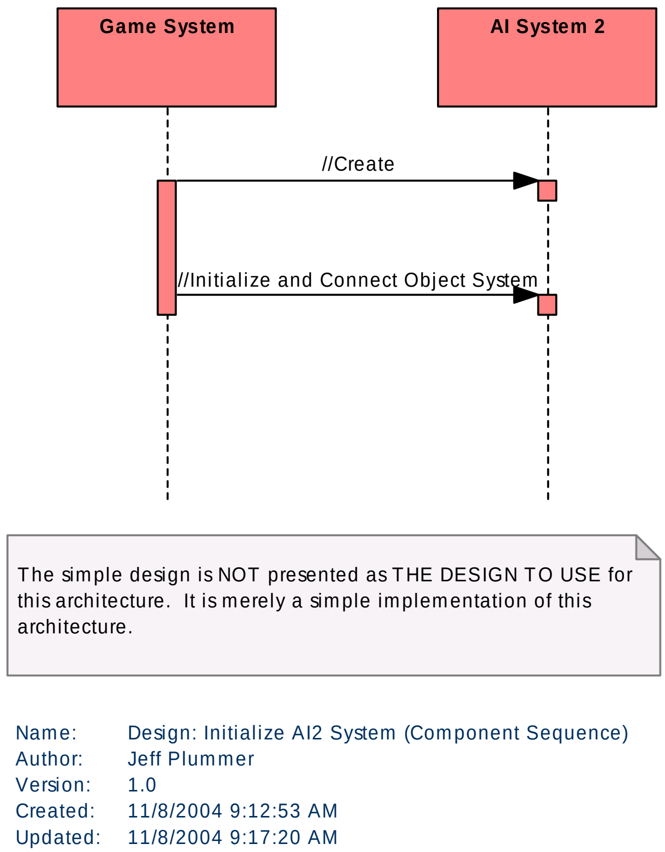


Figure 138 : Initialize

B - 1.2.3.1.1.1.1.1.1 <u>Initialize AI2 System</u>

Type: *public* **<u>UseCase</u>**
Package: Initialize



This diagram shows the sequence of events at the component level required to implement the "Initialize AI2 System" use case.

Figure 139 : Design: Initialize AI2 System (Component Sequence)

Design: Initialize AI2 System (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Create	Game System	AI System 2	Create an instance of the AI2 System.
2	//Initialize and Connec	Game System	AI System 2	Connect the object system, and perform any necessary initialization.

	t Object System			

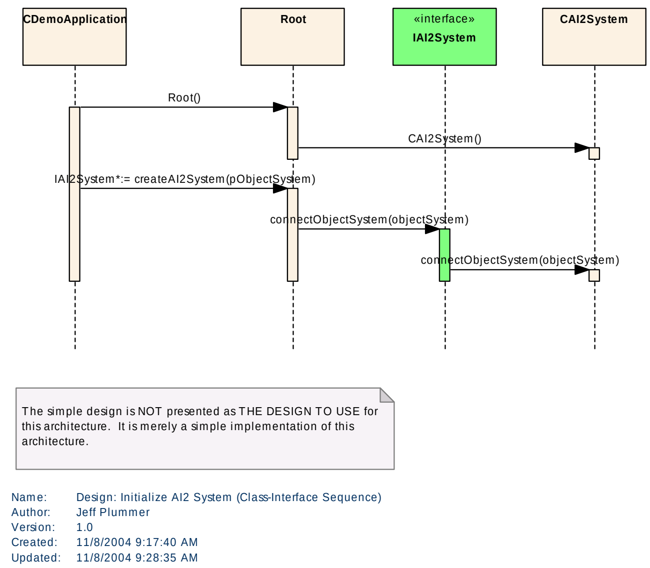


Figure 140 : Design: Initialize AI2 System (Class-Interface Sequence)

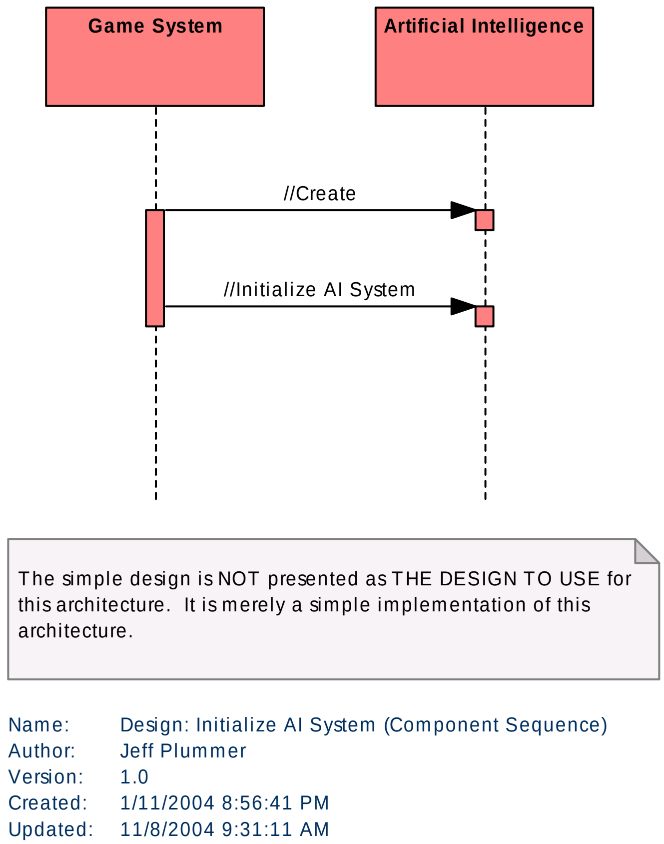
Design: Initialize AI2 System (Class-Interface Sequence) Messages

ID	Message	From Object	To Object	Notes
1	Root()	CDemo Application	Root	Create an instance of the only exported class in the AI system.
2	CAI2System()	Root	CAI2System	The root class in turn creates and AI system object, and returns.
3	createAI2System(IAI2ObjectSystem*)	CDemo Application	Root	Create the AI System by connecting the object component to the AI Component.
4	connect	Root	IAI2Sys	Interface - Connect the

	ObjectSystem(IAI2ObjectSystem*)		tem	object component to the AI component.
5	connectObjectSystem(IAI2ObjectSystem*)	IAI2System	CAI2System	Connect the object component to the AI component.

B - 1.2.3.1.1.1.1.1.2 Initialize AI System

Type: *public* **UseCase**
Package: Initialize

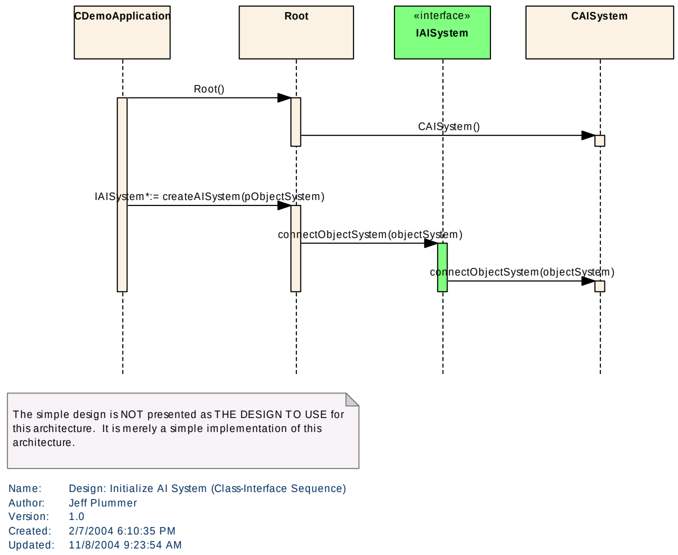


This diagram shows the sequence of events at the component level required to implement the "Initialize AI System" use case.

Figure 141 : Design: Initialize AI System (Component Sequence)

Design: Initialize AI System (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Create	Game System	Artificial Intelligence	Create an instance of the AI system.
2	//Initialize AI System	Game System	Artificial Intelligence	Initialize the AI system and connect it to the object component.



This diagram shows the sequence of events at the class/interface level required to implement the "Initialize AI System" use case.

Figure 142 : Design: Initialize AI System (Class-Interface Sequence)

Design: Initialize AI System (Class-Interface Sequence) Messages

ID	Message	From Object	To Object	Notes
1	Root()	CDemoApplication	Root	Create an instance of the only exported class in the AI system.
2	CAISystem()	Root	CAISystem	The root class in turn creates and AI system object, and returns.
3	createAISystem(IAIObjectSystem*)	CDemoApplication	Root	Create the AI System by connecting the object component to the AI Component.
4	connectObjectSystem(IAIObje	Root	IAISystem	Interface - Connect the object component to the AI component.

	ctSystem*)			
5	connectObjectSystem(IAIObjectSystem*)	IAISystem	CAISystem	Connect the object component to the AI component.

B - 1.2.3.1.1.1.1.1.3 Initialize Graphics 3D System

Type: *public* **UseCase**
Package: Initialize

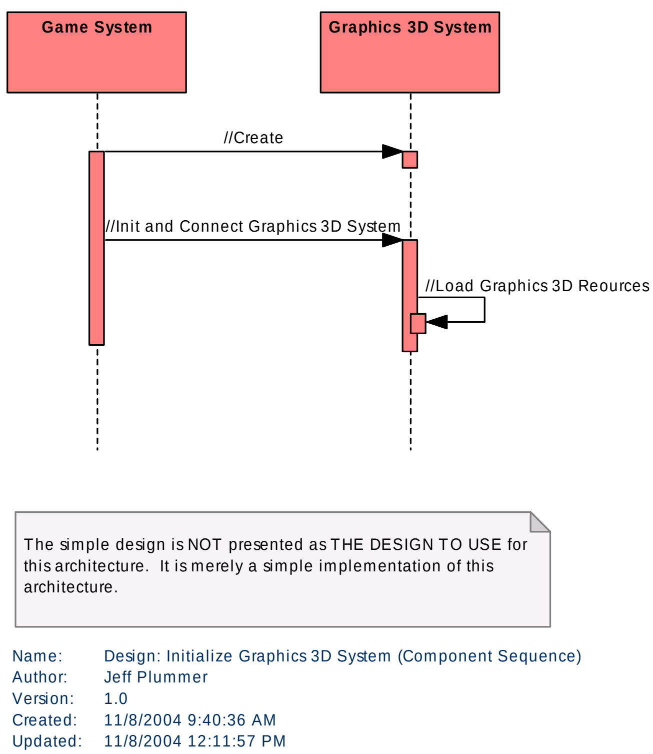


Figure 143 : Design: Initialize Graphics 3D System (Component Sequence)

Design: Initialize Graphics 3D System (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Create	Game System	Graphics 3D System	Create an instance of the graphics 3D system.
2	//Init and Connect Graphics 3D System	Game System	Graphics 3D System	Initialize the graphics 3D system and connect it to the object system.
3	//Load Graphics 3D Reources	Graphics 3D System	Graphics 3D System	Load up any graphics resources you need.

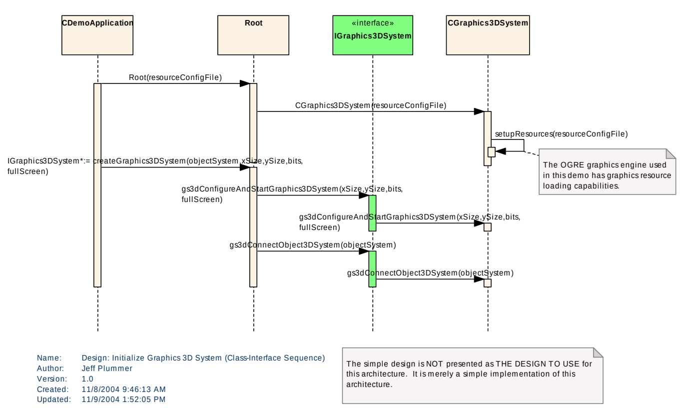


Figure 144 : Design: Initialize Graphics 3D System (Class-Interface Sequence)

Design: Initialize Graphics 3D System (Class-Interface Sequence) Messages

	Messag	From	To	Notes

ID	e	Object	Object	
1	Root(std::string&)	CDemoApplication	Root	Create an instance of the only exported class in the Graphics3D system.
2	CGraphics3DSystem(ProgrammingUtilitiesLibrary::String&)	Root	CGraphics3DSystem	The root class in turn creates and Graphics 3D system object, and returns.
3	setupResources(Programming Utilities Library::String &)	CGraphics3DSystem	CGraphics3DSystem	Load up graphic resource files (meshes, textures, etc.)
4	createGraphics 3DSystem(IGraphics3DObjectSystem *, int, int, int, bool)	CDemoApplication	Root	Create the Graphics3D System by connecting the object component to the Graphics3D Component.
5	gs3dConfigureAndStartGraphics3DSystem(int, int, int, bool)	Root	IGraphics3DSystem	Interface - Setup the graphics window settings.
6	gs3dConfigureAndStartGraphics3DS	IGraphics3DSystem	CGraphics3DSystem	Implementation - Setup the graphics window settings.

	ystem(int, int, int, bool)			
7	gs3dConnectObject3DSystem(IGraphics3DObjectSystem*)	Root	IGraphics3DSystem	Interface - Connect the Object component to the Graphics 3D component.
8	gs3dConnectObject3DSystem(IGraphics3DObjectSystem*)	IGraphics3DSystem	CGraphics3DSystem	Implementation - Connect the Object component to the Graphics 3D component.

B - 1.2.3.1.1.1.1.1.4 Initialize Graphics System

Type: *public* **UseCase**
Package: Initialize

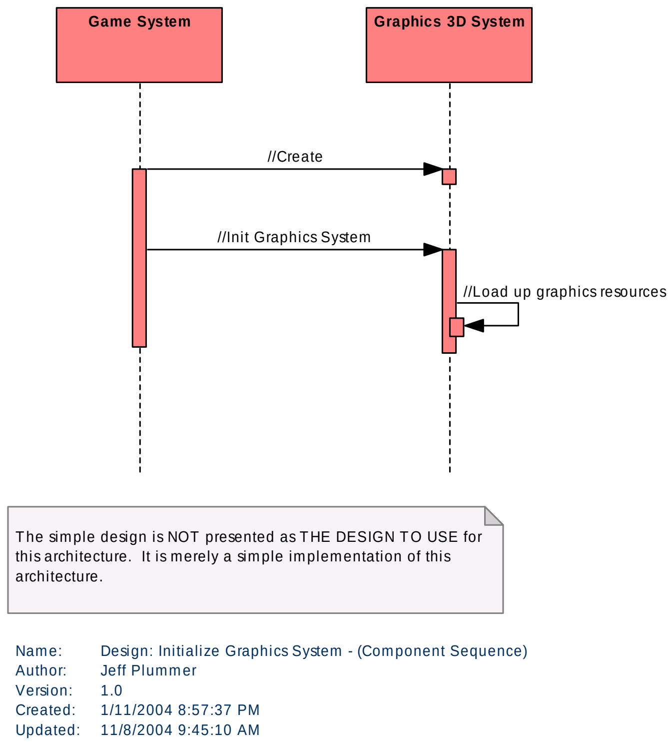


Figure 145 : Design: Initialize Graphics System - (Component Sequence)

Design: Initialize Graphics System - (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Create	Game System	Graphics 3D System	Create an instance of the graphics system.
2	//Init Graphics System	Game System	Graphics 3D System	Initialize the graphics system and connect it to the object system.
3	//Load up graphics resources	Graphics 3D System	Graphics 3D System	Load up any graphics resources you need.

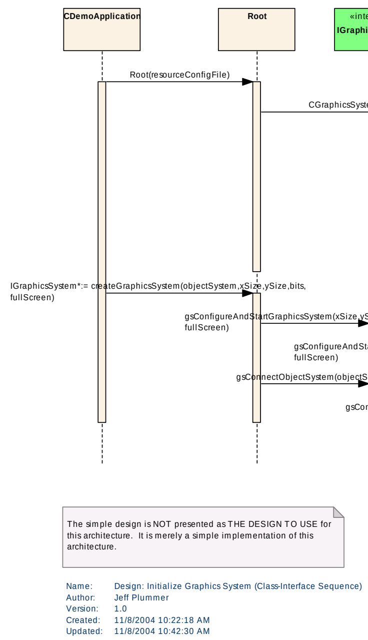


Figure 146 : Design: Initialize Graphics System (Class-Interface Sequence)

Design: Initialize Graphics System (Class-Interface Sequence) Messages

ID	Message	From Object	To Object	Notes
1	Root(std::string&)	CDemoApplication	Root	Create an instance of the only exported class in the Graphics2D system.
2	CGraphicsSystem(ProgrammingUtilitiesLibrary::String&)	Root	CGraphicsSystem	The root class in turn creates and Graphics 2D system object, and returns.
3	CGraphicsReso	CGraphicsSyste	CGraphicsReso	Create an instance of the singleton graphics

	urceMa nager()	m	urceMa nager	resource manager.
4	addArc hiveRes ourceE xtensio n(String &)	CGraph icsSyste m	CGraph icsReso urceMa nager	Based on the config file, add list of file extensions contain graphics resources.
5	addExte nsionIn Archive ToLoad (String &)	CGraph icsSyste m	CGraph icsReso urceMa nager	Based on the config file, add list of file extensions in the resource files are graphics resources.
6	addArc hiveRes ourceL ocation(String&)	CGraph icsSyste m	CGraph icsReso urceMa nager	Based on the config file, add list of directories contain the resource files.
7	loadGra phicsRe sources ()	CGraph icsSyste m	CGraph icsReso urceMa nager	Load the graphics resources
8	createG raphics System(IGraphi csObjec tSystem *, int, int, int, bool)	CDemo Applica tion	Root	Create the graphics system and connect it to the object system.
9	gsConfi gureAn dStartG raphics System(int, int, int, bool)	Root	IGraphi csSyste m	Interface - Configure the graphics system.
1 0	gsConfi gureAn dStartG raphics	IGraphi csSyste m	CGraph icsSyste m	Implementation - Configure the graphics system.

	System(int, int, int, bool)			
11	gsConnectObjectSystem(IGraphicsObjectSystem*)	Root	IGraphicsSystem	Interface - Connect the object system to the graphics system.
12	gsConnectObjectSystem(IGraphicsObjectSystem*)	IGraphicsSystem	CGraphicsSystem	Implementation - Connect the object system to the graphics system.

B - 1.2.3.1.1.1.1.1.5 Initialize Object System

Type: *public* **UseCase**
Package: Initialize

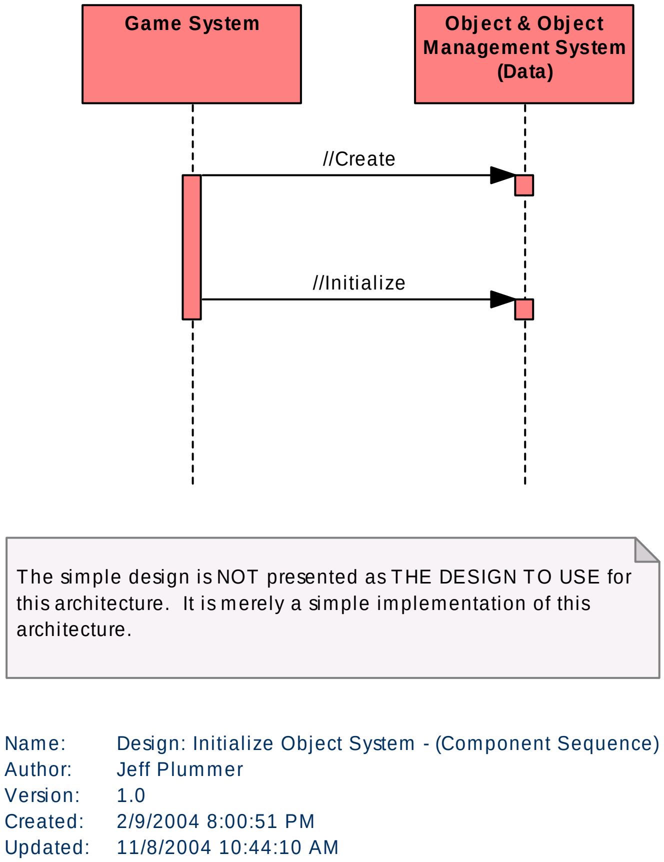


Figure 147 : Design: Initialize Object System - (Component Sequence)

Design: Initialize Object System - (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Create	Game System	Object & Object Management System (Data)	Create an instance of the object system.
2	//Initialize	Game System	Object &	Initialize the object system atleast to the point

			Object Management System (Data)	where the other components can connect to it.

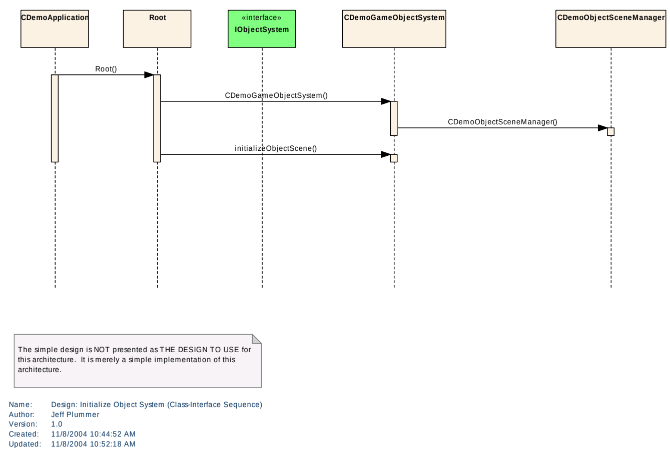


Figure 148 : Design: Initialize Object System (Class-Interface Sequence)

Design: Initialize Object System (Class-Interface Sequence) Messages

ID	Message	From Object	To Object	Notes
1	Root()	CDemoApplication	Root	Create an instance of the only exported class in the Object system.
2	CDemoGameObjectSystem()	Root	CDemoGameObjectSystem	Create an instance of the object system
4	CDemoObjectSceneManager()	CDemoGameObjectSystem	CDemoObjectSceneManager	Create an instance of the only scene manager this demo will use.
5	initializeObject	Root	CDemoGameO	Create some demo objects for us to play

	Scene()		bjectSys tem	around with.

B - 1.2.3.1.1.1.1.1.6 Initialize Game System

Type: *public* **UseCase**
Package: Initialize

Initialize the game system by initializing the object component, and connecting all peripheral components.

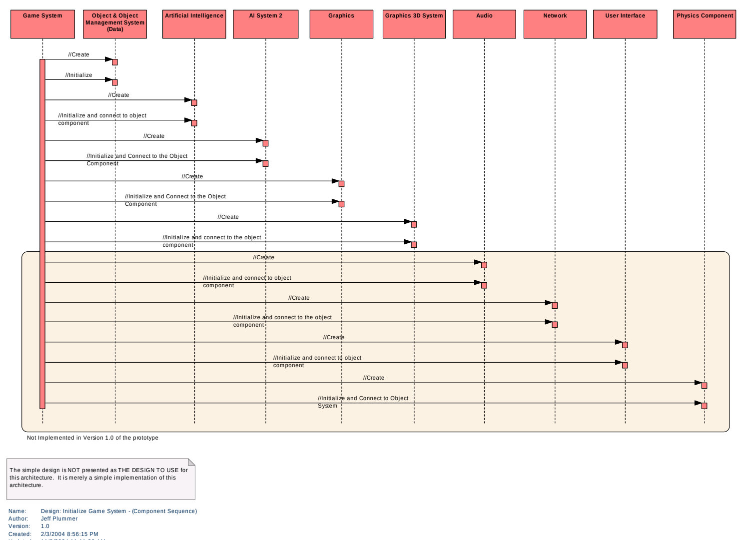


Figure 149 : Design: Initialize Game System - (Component Sequence)

Design: Initialize Game System - (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Create	Game System	Object & Object Management	First create the object component system.

			System (Data)	
2	//Initialize	Game System	Object & Object Management System (Data)	Initialize the component system, atleast to the point where the other components can connect to it.
3	//Create	Game System	Artificial Intelligence	Create an instance of the AI system.
4	//Initialize and connect to object component	Game System	Artificial Intelligence	Connect the AI component to the object component and initialize it so it's ready to handle AI objects.
5	//Create	Game System	AI System 2	Create an instance of the AI2 system.
6	//Initialize and Connect to the Object Component	Game System	AI System 2	Connect the AI2 component to the object component and initialize it so it's ready to handle AI2 objects.
7	//Create	Game System	Graphics	Create an instance of the Graphics 2D system.
8	//Initialize and Connect to the Object Component	Game System	Graphics	Connect the Graphics2D component to the object component and initialize it so it's ready to handle Graphics2D objects.
9	//Create	Game System	Graphics 3D System	Create an instance of the graphics 3D system.
10	//Initialize and connect	Game System	Graphics 3D System	Connect the Graphics3D component to the object system and initialize it so

	to the object component			it's ready to handle graphic3D objects.
11	//Create	Game System	Audio	Create an instance of the audio system.
12	//Initialize and connect to object component	Game System	Audio	Connect the Audio component to the object system and initialize it so it's ready to handle audio objects.
13	//Create	Game System	Network	Create an instance of the networking system.
14	//Initialize and connect to the object component	Game System	Network	Connect the network component to the object systen and initialize it so it's ready to handle network objects.
15	//Create	Game System	User Interface	Create an instance of the UI System.
16	//Initialize and connect to object component	Game System	User Interface	Connect the UI component to the object system and initialize it so it's ready to handle UI objects.
17	//Create	Game System	Physics Component	Create an instance of the Physics system.
18	//Initialize and Connect to Object System	Game System	Physics Component	Connect the Physics component to the object component and initialize it so it's ready to handle Physics objects.

B - 1.2.3.2 Tick

This diagram shows the use cases involved in ticking the prototype.

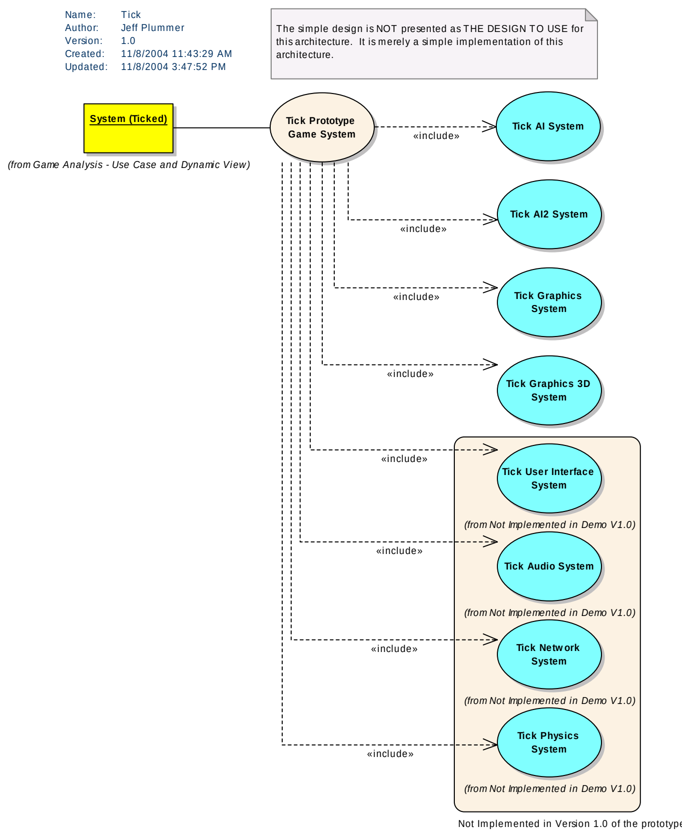


Figure 150 : Tick

B - 1.2.3.2.1.1.1.1.1 <u>Tick AI System</u>

Type: *public* **<u>UseCase</u>**
Package: Tick

Tick the artificial intelligence component. Causes objects to bounce around the screen. Not a very complex AI system.

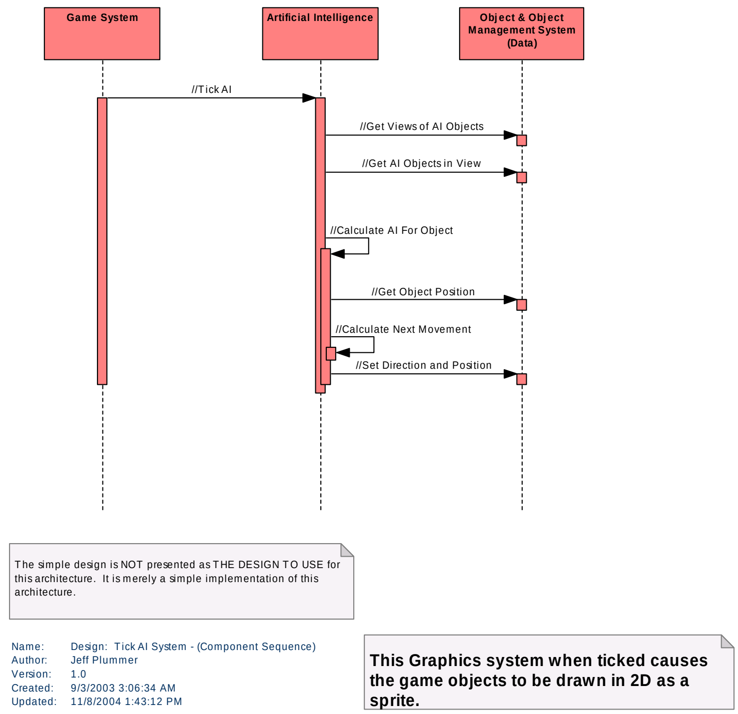


Figure 151 : Design: Tick AI System - (Component Sequence)

Design: Tick AI System - (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick AI	Game System	Artificial Intelligence	Tick the AI Component
2	//Get Views of AI Objects	Artificial Intelligence	Object & Object Management System (Data)	Get an list of AI views to process. Views contain some context, and a list of objects.
3	//Get AI Objects in View	Artificial Intellige	Object & Object	Get the list of AI processable objects in the view.

		nce	Management System (Data)	
4	//Calculate AI For Object	Artificial Intelligence	Artificial Intelligence	The AI component will then calculate the behavior of the object it is going to process.
5	//Get Object Position	Artificial Intelligence	Object & Object Management System (Data)	Get the Position of the object
6	//Calculate Next Movement	Artificial Intelligence	Artificial Intelligence	Based on it's current position, and it's movement direction, calculate it's next move.
7	//Set Direction and Position	Artificial Intelligence	Object & Object Management System (Data)	Write the position info back into the object.

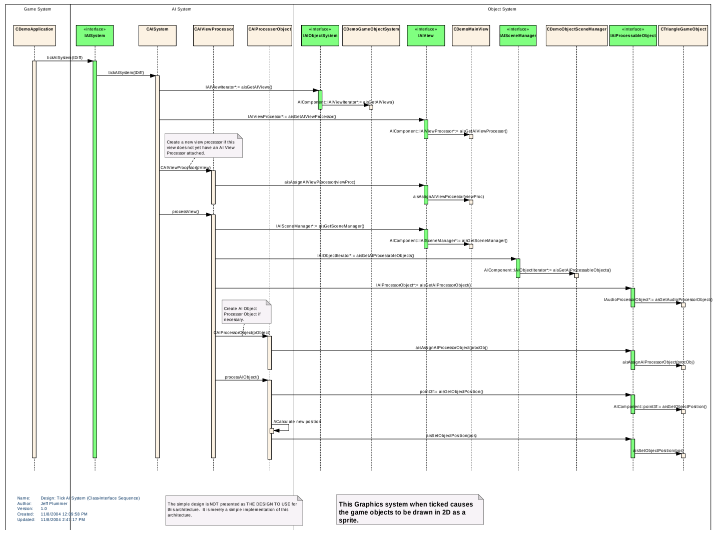


Figure 152 : Design: Tick AI System (Class-Interface Sequence)

Design: Tick AI System (Class-Interface Sequence) Messages

ID	Message	From Object	To Object	Notes
1	tickAISystem(float)	CDemoApplication	IAISystem	Interface - Tick the AI system.
2	tickAISystem(float)	IAISystem	CAISystem	Implementation - Tick the AI System.
3	aisGetAIViews()	CAISystem	IAIObjectSystem	Interface - Get Views of AI objects to process... This prototype only contains one view.
4	aisGetAIViews()	IAIObjectSystem	CDemoGameObjectSystem	Implementation - Get Views of AI objects to process... This prototype only contains one view.
5	aisGetAIViewProcessor()	CAISystem	IAIView	Interface - Get the AI View Processor if it exists.
6	aisGetAIViewP	IAIView	CDemoMainVi	Implementation - Get the AI View Processor if it

	rocessor()		ew	exists.
7	CAIViewProcessor(IAIView*)	CAISystem	CAIViewProcessor	Create a view processor if this view does not yet have one - i.e. this is our first time processing this view.
8	aisAssignAIViewProcessor(IAIViewProcessor*)	CAIViewProcessor	IAIView	Interface - Assign the view processor to the view.
9	aisAssignAIViewProcessor(AIComponent::IAIViewProcessor*)	IAIView	CDemoMainView	Implementation - Assign the view processor to the view.
10	processView()	CAISystem	CAIViewProcessor	AI Process the view
11	aisGetSceneManager()	CAIViewProcessor	IAIView	Interface - Get the Scenemanager (structured list of objects to process)
12	aisGetSceneManager()	IAIView	CDemoMainView	Implementation - Get the Scenemanager (structured list of objects to process)
13	aisGetAIProcessableObjects()	CAIViewProcessor	IAISceneManager	Interface - Get Ordered list of objects to process.
14	aisGetAIProcessableObjects()	IAISceneManager	CDemoObjectSceneManager	Implementation - Get Ordered list of objects to process.
15	aisGetAIProcessorObject()	CAIViewProcessor	IAIProcessableObject	Interface - Get the AI object processor responsible for processing this object.
1	asGetA	IAIProc	CTriang	Implementation - Get the

6	udioProcessorObject()	essableObject	leGameObject	AI object processor responsible for processing this object.
17	CAIProcessorObject(IAIProcessableObject*)	CAIViewProcessor	CAIProcessorObject	Create AI Object Processor Object if necessary.
18	aisAssignAIProcessorObject(IAIProcessorObject*)	CAIProcessorObject	IAIProcessableObject	Interface - Assign the processor object to the game object.
19	aisAssignAIProcessorObject(IAIProcessorObject*)	IAIProcessableObject	CTriangleGameObject	Implementation - Assign the processor object to the game object.
20	processAIObject()	CAIViewProcessor	CAIProcessorObject	Perform AI Processing on this object
21	aisGetObjectPosition()	CAIProcessorObject	IAIProcessableObject	Interface - Get the game object's position.
22	aisGetObjectPosition()	IAIProcessableObject	CTriangleGameObject	Implementation - Get the game object's position.
23	//Calculate new position	CAIProcessorObject	CAIProcessorObject	
24	aisSetObjectPosition(point3f&)	CAIProcessorObject	IAIProcessableObject	Interface - Set the game object's new position
25	aisSetObjectPosition(AICompo	IAIProcessableObject	CTriangleGameObject	Implementation - Set the game object's new position

	nent::point3f&)			

B - 1.2.3.2.1.1.1.1.2 Tick AI2 System

Type: *public* **UseCase**
Package: Tick

Tick the artificial intelligence component. Causes objects to rotate. Not a very complex AI system.

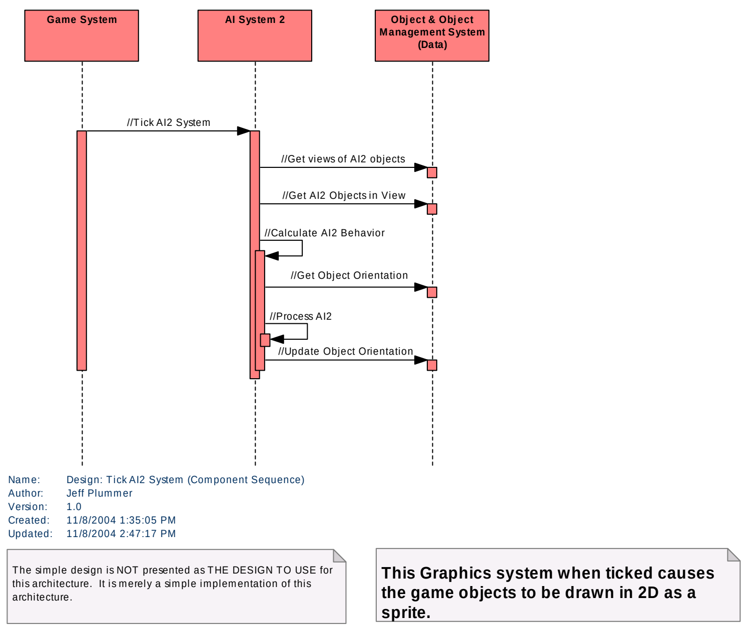


Figure 153 : Design: Tick AI2 System (Component Sequence)

Design: Tick AI2 System (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick AI2 System	Game System	AI System 2	Tick the AI2 Component

2	//Get views of AI2 objects	AI System 2	Object & Object Management System (Data)	Get an list of AI views to process. Views contain some context, and a list of objects.
3	//Get AI2 Objects in View	AI System 2	Object & Object Management System (Data)	Get the list of AI2 processable objects in the view.
4	//Calculate AI2 Behavior	AI System 2	AI System 2	The AI2 component will then calculate the behavior of the object it is going to process.
5	//Get Object Orientation	AI System 2	Object & Object Management System (Data)	Get the objects orientation data.
6	//Process AI2	AI System 2	AI System 2	Rotate the object
7	//Update Object Orientation	AI System 2	Object & Object Management System (Data)	Update the object's orientation using the new data.

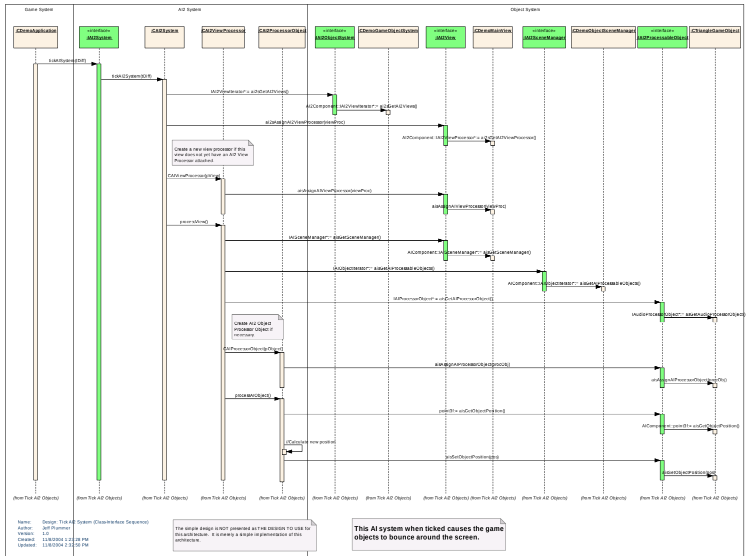


Figure 154 : Design: Tick AI2 System (Class-Interface Sequence)

Design: Tick AI2 System (Class-Interface Sequence) Messages

ID	Message	From Object	To Object	Notes
1	tickAISystem(float)			Interface - Tick the AI system.
2	tickAI2System(float)			Implementation - Tick the AI2 System.
3	ai2sGetAI2Views()			Interface - Get Views of AI2 objects to process... This prototype only contains one view.
4	ai2sGetAI2Views()			Implementation - Get Views of AI2 objects to process... This prototype only contains one view.
5	ai2sAssignAI2ViewProcessor(IAI2Vi			Interface - Get the AI2 View Processor if it exists.

	ewProcessor*)			
6	ai2sGetAI2ViewProcessor()			Implementation - Get the AI2 View Processor if it exists.
7	CAIViewProcessor(IAIView*)			Create a view processor if this view does not yet have one - i.e. this is our first time processing this view.
8	aisAssignAIViewProcessor(IAIViewProcessor*)			Interface - Assign the view processor to the view.
9	aisAssignAIViewProcessor(AIComponent::IAIViewProcessor*)			Implementation - Assign the view processor to the view.
10	processView()			AI Process the view
11	aisGetSceneManager()			Interface - Get the Scenemanager (structured list of objects to process)
12	aisGetSceneManager()			Implementation - Get the Scenemanager (structured list of objects to process)
13	aisGetAIProcessableObjects()			Interface - Get Ordered list of objects to process.
14	aisGetAIProcessableObjects()			Implementation - Get Ordered list of objects to process.
15	aisGetAIProces			Interface - Get the AI object processor

	sorObject()			responsible for processing this object.
16	asGetAudioProcessorObject()			Implementation - Get the AI object processor responsible for processing this object.
17	CAIProcessorObject(IAIProcessableObject*)			Create AI Object Processor Object if necessary.
18	aisAssignAIProcessorObject(IAIProcessorObject*)			Interface - Assign the processor object to the game object.
19	aisAssignAIProcessorObject(IAIProcessorObject*)			Implementation - Assign the processor object to the game object.
20	processAIObject()			Perform AI Processing on this object
21	aisGetObjectPosition()			Interface - Get the game object's position.
22	aisGetObjectPosition()			Implementation - Get the game object's position.
23	//Calculate new position			
24	aisSetObjectPosition(point3f&)			Interface - Set the game object's new position
2	aisSetO			Implementation - Set the

5	bjectPosition(AIComponent::point3f&)			game object's new position

B - 1.2.3.2.1.1.1.1.3 Tick Graphics 3D System

Type: *public* **UseCase**
Package: Tick

Draws objects as 3D objects

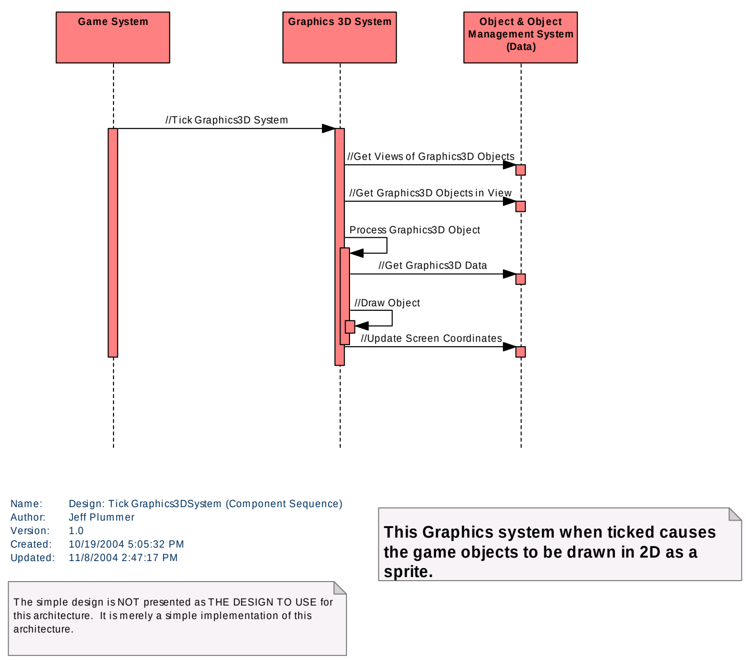


Figure 155 : Design: Tick Graphics3DSystem (Component Sequence)

Design: Tick Graphics3DSystem (Component Sequence) Messages

ID	Message	From Object	To Object	Notes

1	//Tick Graphics3D System	Game System	Graphics 3D System	Tick the Graphics3D Component.
2	//Get Views of Graphics3D Objects	Graphics 3D System	Object & Object Management System (Data)	Get an list of Graphics3D views to process. Views contain some context, and a list of objects.
3	//Get Graphics3D Objects in View	Graphics 3D System	Object & Object Management System (Data)	Get the list of Graphics3D processable objects in the view.
4	Process Graphics3D Object	Graphics 3D System	Graphics 3D System	
5	//Get Graphics3D Data	Graphics 3D System	Object & Object Management System (Data)	Get data like position, graphics resources, etc. to draw.
6	//Draw Object	Graphics 3D System	Graphics 3D System	
7	//Update Screen Coordinates	Graphics 3D System	Object & Object Management System (Data)	The graphics3D engine updates screen coordinate data in case other componentes use that data.

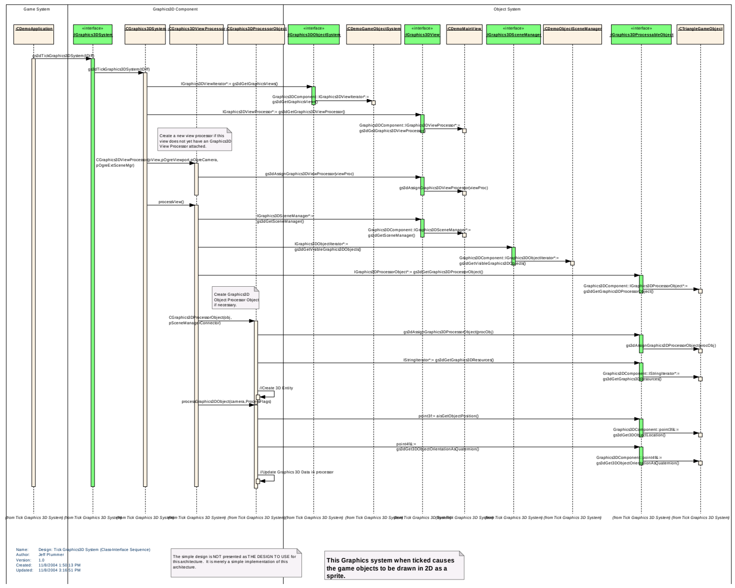


Figure 156 : Design: Tick Graphics3D System (Class-Interface Sequence)

Design: Tick Graphics3D System (Class-Interface Sequence) Messages

ID	Message	From Object	To Object	Notes
1	gs3dTickGraphics3DSystem(float)			Interface - Tick the Graphics3D system.
2	gs3dTickGraphics3DSystem(float)			Implementation - Tick the Graphics3D System.
3	gs3dGetGraphicsViews()			Interface - Get Views of Graphics3D objects to process... This prototype only contains one view.
4	gs3dGetGraphicsViews()			Implementation - Get Views of Graphics3D objects to process... This prototype only contains

				one view.
5	gs3dGetGraphics3DViewProcessor()			Interface - Get the Graphics3D View Processor if it exists.
6	gs3dGetGraphics3DViewProcessor()			Implementation - Get the Graphics3D View Processor if it exists.
7	CGraphics3DViewProcessor(IGraphics3DView*, Ogre::Viewport*, Ogre::Camera*, Ogre::CExternalSceneManager*)			Create a view processor if this view does not yet have one - i.e. this is our first time processing this view.
8	gs3dAssignGraphics3DViewProcessor(IGraphics3DViewProcessor*)			Interface - Assign the view processor to the view.
9	gs3dAssignGraphics3DViewProcessor(Graphics3D			Implementation - Assign the view processor to the view.

	Component::IGraphics3DViewProcessor*)			
10	processView()			Graphics3D Process the view
11	gs3dGetSceneManager()			Interface - Get the Scenemanager (structured list of objects to process)
12	gs3dGetSceneManager()			Implementation - Get the Scenemanager (structured list of objects to process)
13	gs3dGetVisibleGraphics3DObjects()			Interface - Get Ordered list of objects to process.
14	gs3dGetVisibleGraphics3DObjects()			Implementation - Get Ordered list of objects to process.
15	gs3dGetGraphics3DProcessorObject()			Interface - Get the Graphics3D object processor responsible for processing this object.
16	gs3dGetGraphics3DProcessorObject()			Implementation - Get the Graphics3D object processor responsible for processing this object.
17	CGraphics3DProcessorObject(IGraphics3DProcessab			Create Graphics3D Object Processor Object if necessary.

	leObject*, Ogre::CExternalSceneManager*)			
18	gs3dAssignGraphics3DProcessorObject(IGraphics3DProcessorObject*)			Interface - Assign the processor object to the game object.
19	gs3dAssignGraphics3DProcessorObject(Graphics3DComponent::IGraphics3DProcessorObject*)			Implementation - Assign the processor object to the game object.
20	gs3dGetGraphics3DResources()			Interface - Get the Graphics3D Resource information required to draw the object in 3D.
21	gs3dGetGraphics3DResources()			Implementation - Get the Graphics3D Resource information required to draw the object in 3D.
22	//Create 3D Entity			Create the entity using OGREs resource manager.
23	process Graphic			Perform Graphics3D Processing on this object

	s3DObject(IGraphics3DCamera*, unsigned int)			
24	aisGetObjectPosition()			Interface - Get the game object's position.
25	gs3dGet3DObjectLocation()			Implementation - Get the game object's position.
26	gs3dGet3DObjectOrientationAsQuaternion()			Interface - Get the game object's orientation
27	gs3dGet3DObjectOrientationAsQuaternion()			Implementation - Get the game object's orientation
28	//Update Graphics 3D Data in processor			Update the OGRE graphics entity in the processor.

B - 1.2.3.2.1.1.1.1.4 Tick Graphics System

Type: *public* **UseCase**
Package: Tick

Draws objects as 2D sprite

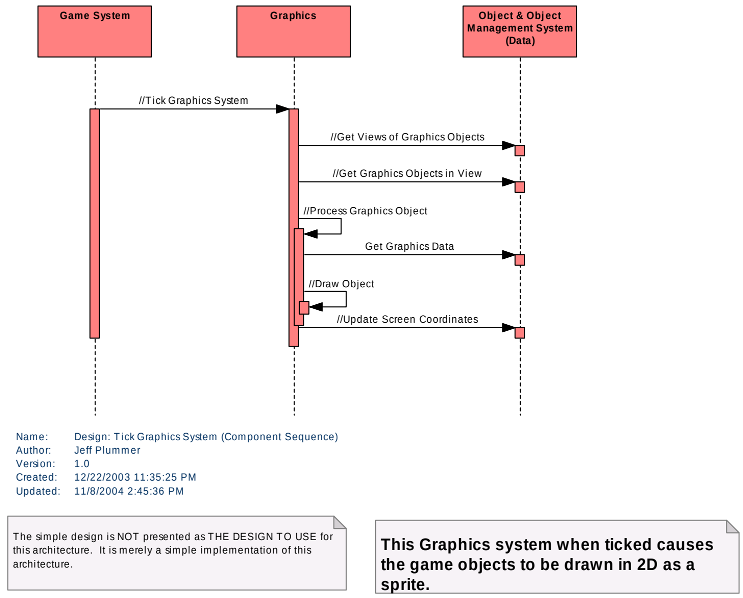


Figure 157 : Design: Tick Graphics System (Component Sequence)

Design: Tick Graphics System (Component Sequence) Messages

ID	Message	From Object	To Object	Notes
1	//Tick Graphics System	Game System	Graphics	Tick the 2D graphics component.
2	//Get Views of Graphics Objects	Graphics	Object & Object Management System (Data)	Get an list of Graphics views to process. Views contain some context, and a list of objects.
3	//Get Graphics Objects in View	Graphics	Object & Object Management System (Data)	Get the list of Graphics processable objects in the view.

4	//Process Graphics Object	Graphics	Graphics	
5	Get Graphics Data	Graphics	Object & Object Management System (Data)	Get data like position, graphics resources, etc. to draw.
6	//Draw Object	Graphics	Graphics	
7	//Update Screen Coordinates	Graphics	Object & Object Management System (Data)	The graphics engine updates screen coordinate data in case other componentes use that data.

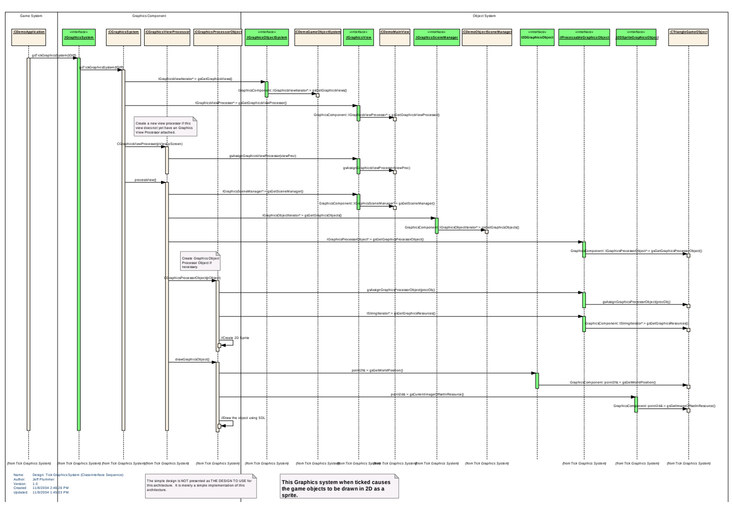


Figure 158 : Design: Tick Graphics System (Class-Interface Sequence)

Design: Tick Graphics System (Class-Interface Sequence) Messages

ID	Message	From Object	To Object	Notes
1	gsTickGraphicsSystem(float)			Interface - Tick the Graphics system.
2	gsTickGraphicsSystem(float)			Implementation - Tick the Graphics System.
3	gsGetGraphicsViews()			Interface - Get Views of Graphics objects to process... This prototype only contains one view.
4	gsGetGraphicsViews()			Implementation - Get Views of Graphics objects to process... This prototype only contains one view.
5	gsGetGraphicsViewProcessor()			Interface - Get the Graphics View Processor if it exists.
6	gsGetGraphicsViewProcessor()			Implementation - Get the Graphics View Processor if it exists.
7	CGraphicsViewProcessor(IGraphicsView*, SDL_Surface*)			Create a view processor if this view does not yet have one - i.e. this is our first time processing this view.
8	gsAssignGraphicsViewProcessor(IGraphicsVi			Interface - Assign the view processor to the view.

	ewProcessor*)			
9	gsAssignGraphicsViewProcessor(GraphicsComponent::IGraphicsViewProcessor*)			Implementation - Assign the view processor to the view.
10	processView()			Graphics Process the view
11	gsGetSceneManager()			Interface - Get the Scenemanager (structured list of objects to process)
12	gsGetSceneManager()			Implementation - Get the Scenemanager (structured list of objects to process)
13	gsGetGraphicsObjects()			Interface - Get Ordered list of objects to process.
14	gsGetGraphicsObjects()			Implementation - Get Ordered list of objects to process.
15	gsGetGraphicsProcessorObject()			Interface - Get the Graphics object processor responsible for processing this object.
16	gsGetGraphicsProcessorObject()			Implementation - Get the Graphics object processor responsible for processing this object.
17	CGraphicsProcessorObject(IProcessableGraph			Create Graphics Object Processor Object if necessary.

	icsObject*)			
18	gsAssignGraphicsProcessorObject(IGraphicsProcessorObject*)			Interface - Assign the processor object to the game object.
19	gsAssignGraphicsProcessorObject(GraphicsComponent::IGraphicsProcessorObject*)			Implementation - Assign the processor object to the game object.
20	gsGetGraphicsResources()			Interface - Get the Graphics Resource information required to draw the object in 2D.
21	gsGetGraphicsResources()			Implementation - Get the Graphics Resource information required to draw the object in 2D.
22	//Create 2D Sprite			Create the entity using SDL to manage sprites.
23	drawGraphicsObject()			Perform Graphics Processing on this object
24	gsGetWorldPosition()		I2DGraphicsObject	Get the position of the 2D object
25	gsGetWorldPosition()	I2DGraphicsObject		Get the 2D objects position in the world
26	gsCurrentImageOffsetI			Interface - Get the sprite offset in the 2D image

	nResource()			
27	gsGetImageOffsetInResource()			Implementation - Get the sprite offset in the 2D image
28	//Draw the object using SDL			Use SDL to blit the sprite

B - 1.2.3.2.1.1.1.1.5 Tick Prototype Game System

Type: *public* **UseCase**
Package: Tick

This design dependent use case represents the process of ticking all the domain-specific components to create the game behavior.

B - 1.2.4 Component View

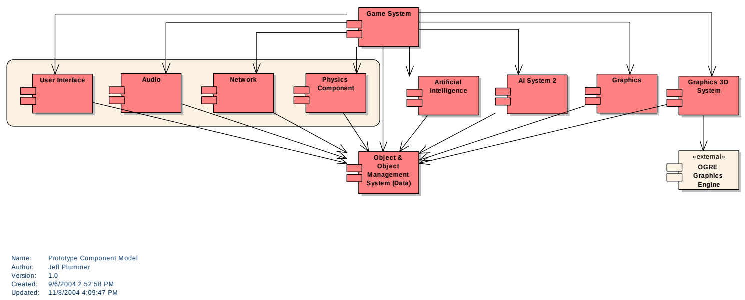


Figure 159 : Prototype Component Model

B - 1.2.4.1.1.1.1.1.1 AI System 2

Type: *public* **Component**
Package: Component View

This component is the AI2 System DLL that when attached to the object component performs rotation AI on the objects.

B - 1.2.4.1.1.1.1.1.2 Artificial Intelligence

Type: *public* **Component**
Package: Component View

This component is the AI System DLL that when attached to the object component performs movement AI on the objects.

B - 1.2.4.1.1.1.1.1.3 Audio

Type: *public* **Component**
Package: Component View

This component is the Audio System DLL that when attached to the object component performs sound processing on the objects.

B - 1.2.4.1.1.1.1.1.4 Game System

Type: *public* **Component**
Package: Component View

Represents the master game system EXE file.

B - 1.2.4.1.1.1.1.1.5 Graphics

Type: *public* **Component**
Package: Component View

This component is the Graphics System DLL that when attached to the object component draws the objects in 2D.

B - 1.2.4.1.1.1.1.1.6 Graphics 3D System

Type: *public* **Component**
Implements: *IGraphics3DSystem.*
Package: Component View

This component is the Graphics System DLL that when attached to the object component draws the objects in 3D.

B - 1.2.4.1.1.1.1.1.7 Network

Type: *public* **Component**
Package: Component View

This component is the Network System DLL that when attached to the object component performs network processing on the objects.

B - 1.2.4.1.1.1.1.1.8 Object & Object Management System (Data)

Type: *public* **Component**
Implements: *IGraphics3DObjectSystem.*
Package: Component View

The Game Objects and Object Management System.

B - 1.2.4.1.1.1.1.1.9 OGRE Graphics Engine

Type: *public «external»* **Component**
Package: Component View

The OGRE (www.ogre3d.org) graphics engine was used in the prototype, and actually provides some proof that it is not difficult to integrate an existing graphics engine into this architecture.

B - 1.2.4.1.1.1.1.1.10 Physics Component

Type: *public* **Component**
Package: Component View

This component is the Physics System DLL that when attached to the object component performs physics calculations on the objects.

B - 1.2.4.1.1.1.1.1.11 User Interface

Type: *public* **Component**
Package: Component View

This component is the User Interface System DLL that when attached to the object component allows UI listening objects to exist.