First Turkish Symposium on Model-Driven Software Development
(TMODELS – 2009)

20 May 2009
Bilkent University, Ankara, Turkey

Model-driven Approach for Board Game Development

Group Members:

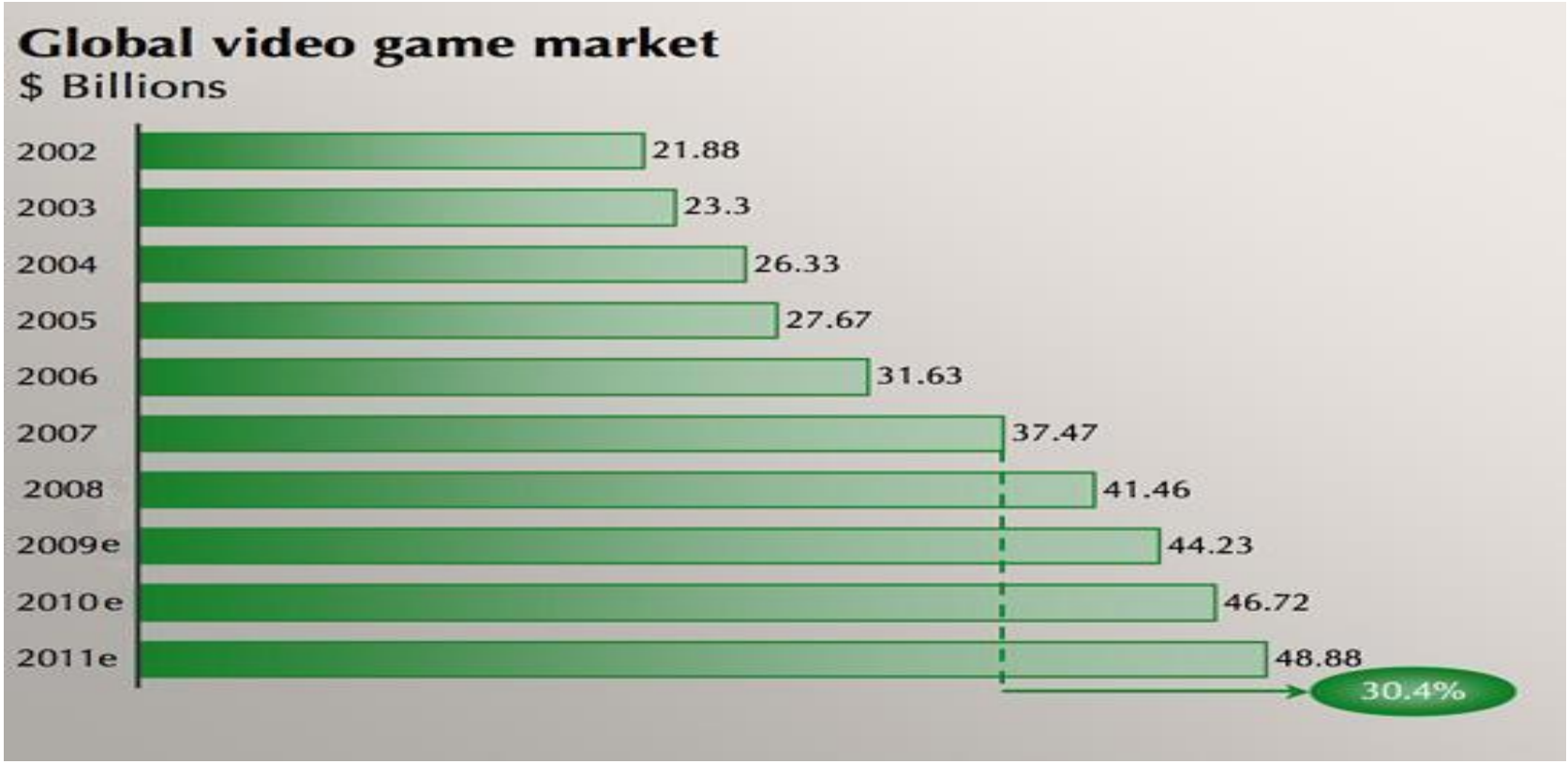
- Doğan ALTUNBAY, Bilkent University
- M. Gökhan METİN, Bilkent University
- M. Eser ÇETİNKAYA, Bilkent University & Havelsan

Agenda

- Introduction & Motivation
- Domain Description
- Mapping Domain Concepts to Grammar
- Metamodel Definition
- Static Semantics
- UML Profiling
- Sample Models
- Model to Text Transformation
- Model to Model Transformation
- Conclusion

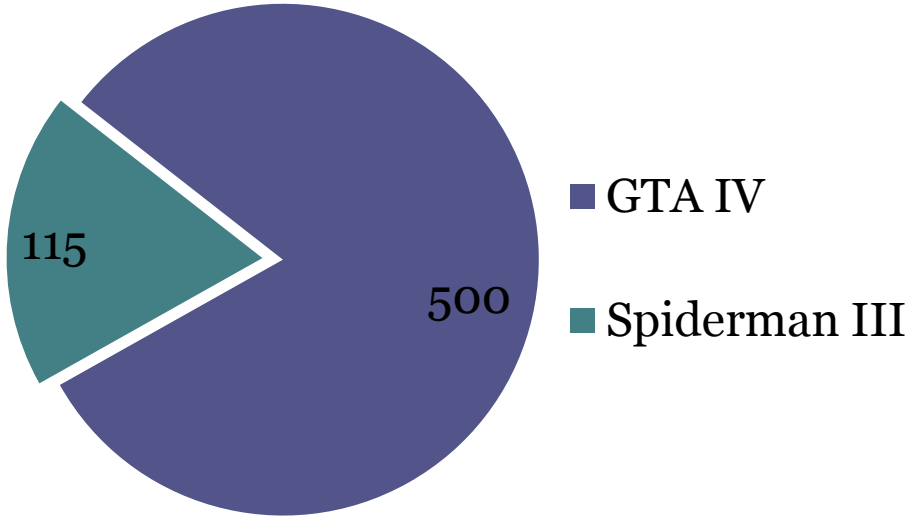
Introduction

- Game Industry has grown to mainstream market



GTA IV vs Spiderman III

- *Grand Theft Auto* IV, which took in over USD$500 million in sales during its opening week.
- The movie Spiderman III could only reach to USD$115 million in first week gross.



Challenge for Game Companies

- Software quality is becoming of utmost importance
- For large software systems it may take quite a lot of human time and energy to gather requirements
- People have increasing demands on technology, requirements keep changing
- Software systems need to be flexibly adaptable to changing requirements

New Way in Game Development

- Raising the abstraction level from the "**solution domain**" to the "**problem domain**" and using the model-driven based approaches.
- Domain specific languages(DSL) may become increasingly better assembly lines.
- New approach brings noticeable advantage to companies in Game market.

Why we concentrate on Board Games

- Games could have totally different domains

- Too generic
- Not practical
- Unnecessarily complicated

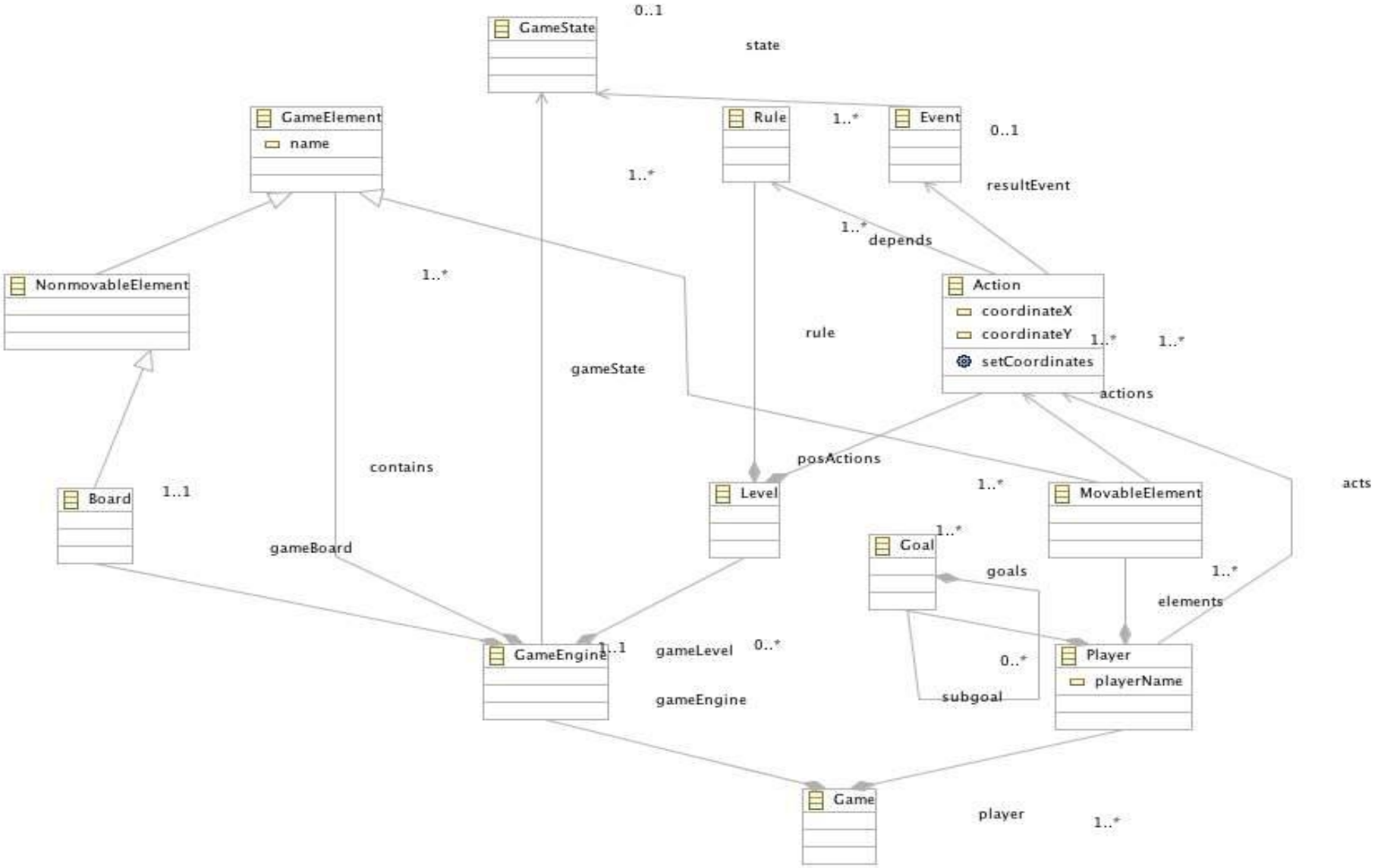
Domain Description

- ***Game:*** A Game is a root class which consists of Player(s) and GameEngine
- ***GameEngine:*** A game must have a GameEngine which is responsible for running the game based on the defined rules of the game.
- ***GameElement:*** The GameElement represents all the objects inside a specific game.
- ***Player:*** Players are the decision makers inside a game.
- ***Event:*** Event is a condition in which the opponents actions are restricted.
- ***Action:*** Actions are the movements of Players.
- ***GameState:*** GameState metaclass represents the current condition of the game at any instance.
- ***Goal:*** In any board game, goal is the state which players try to attain by creating actions on GameElements.
- ***Sub Goal:*** Sub-Goal concept is defined such that in some games it is required to achieve global goal by completing its parts in order.

Domain Description

- ***NonMovableElement:*** NonmovableElements are the ones which cannot be manipulated by any player by an action.
- ***MovableElement:*** *MovableElements are the ones which can be manipulated by any player*.
- ***Rule:*** R*ules are the constraints that define how to setup a system before playing, relationship between the game and the player*.
- ***Board:*** Board is a surface on which MovableElements are located.

Metamodel Definition



DSL Grammar

<Game> :: = <GameEngine> , <Rules> , <Player>;

<Rules> ::= <text>

<GameEngine> ::= <GameElement> , <Rules> , <Level> , <Goal> |
<GameElement> , <Rules> , <Goal>;

<GameElement> ::= <MovableElement> , <NonmovableElement>;

<MovableElement> ::= <Token> , <Action> , <VisibleElement> | <Token>
<Action> , <InvisibleElement>;

<VisibleElement> ::= <Token>;

<InvisibleElement> ::= <Timer>;

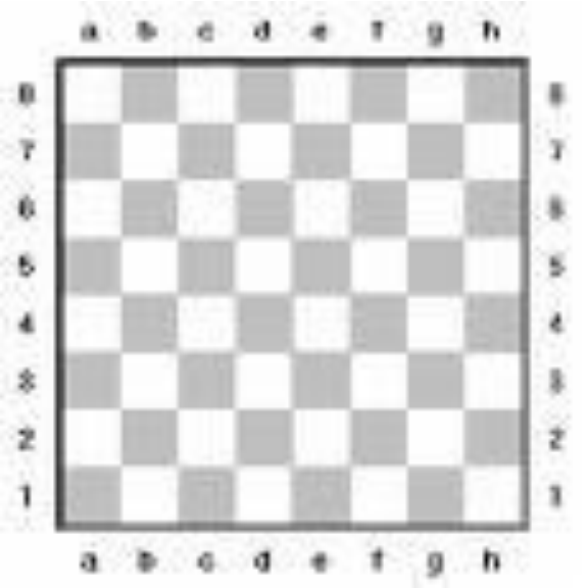
<NonmovableElement> ::= <Board> , <Player> , <ScoreBoard>;

<Player> ::= <GameState> , <MovableElement> , <Goal>;

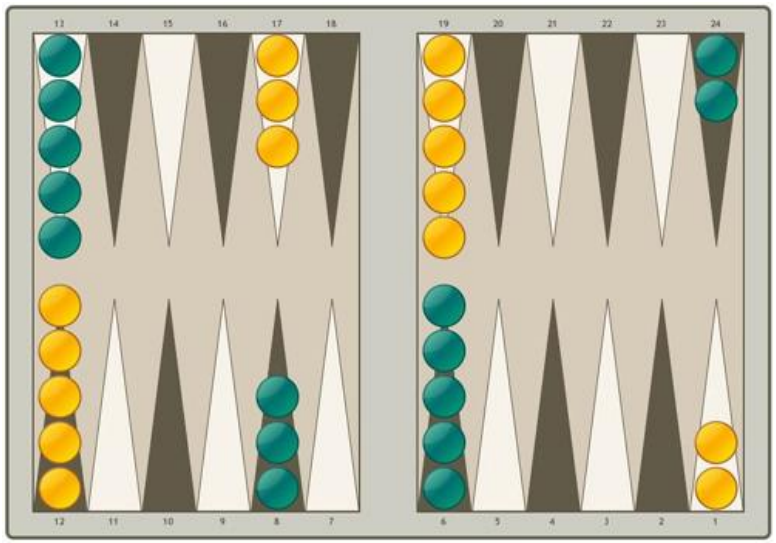
Static Semantics

- context GameEngine inv:not self.Board.oclIsUndefined &
  self.Board.size() = 1
- context Player inv:self.movableElement.size() >= 1
- context GameEngine inv: level.size() >= 1
- context Goal inv:self.reject(g | self.subgoals.exists(self = g))
- context Level inv:self.reject(g | self.rules = g.rules)

Concrete Syntax



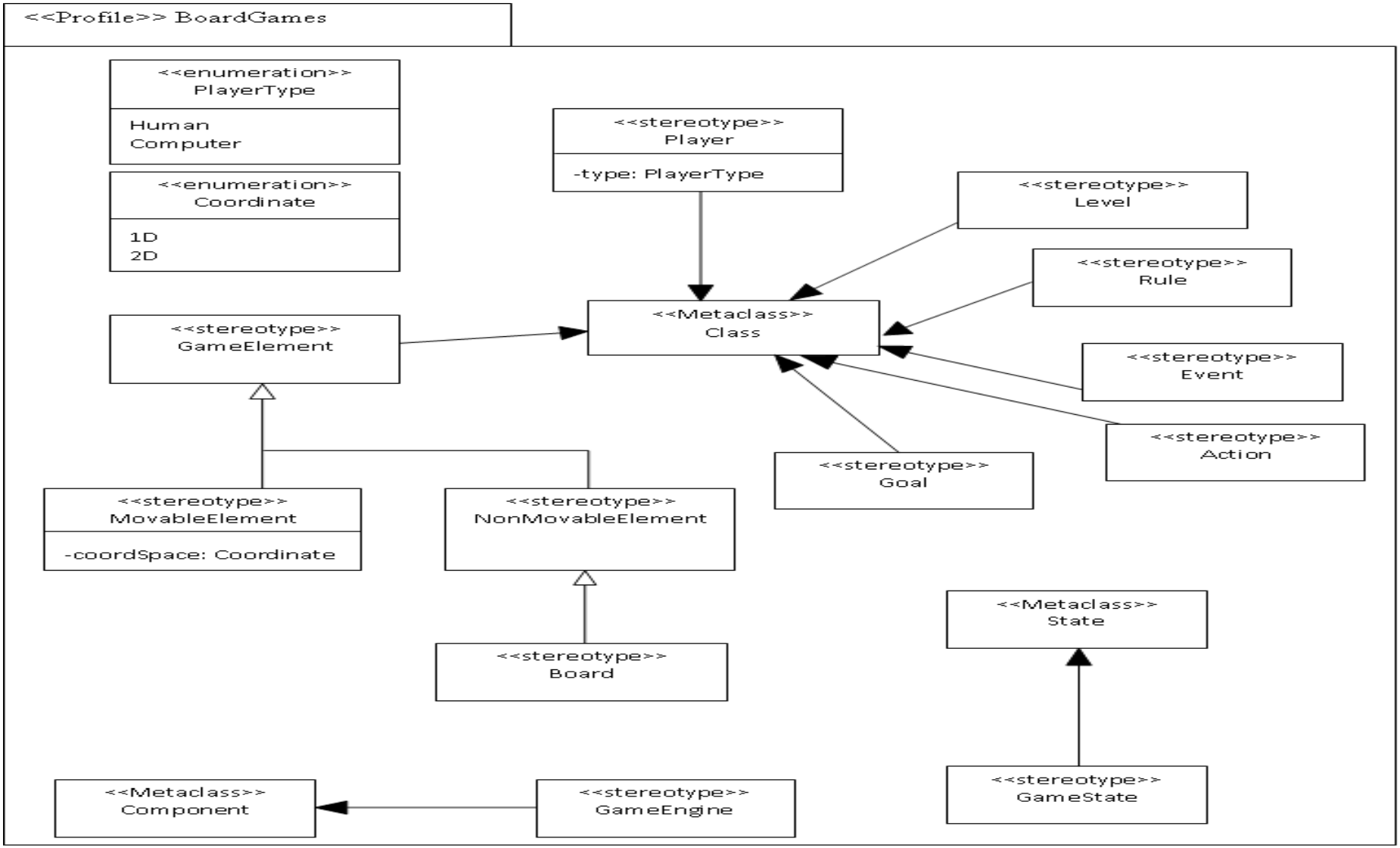
飛	桂	角	妃	玉	角	桂	飛
歩	歩	歩	歩	歩	歩	歩	歩
歩	歩	歩	歩	歩	歩	歩	歩
飛	桂	角	妃	王	角	桂	飛



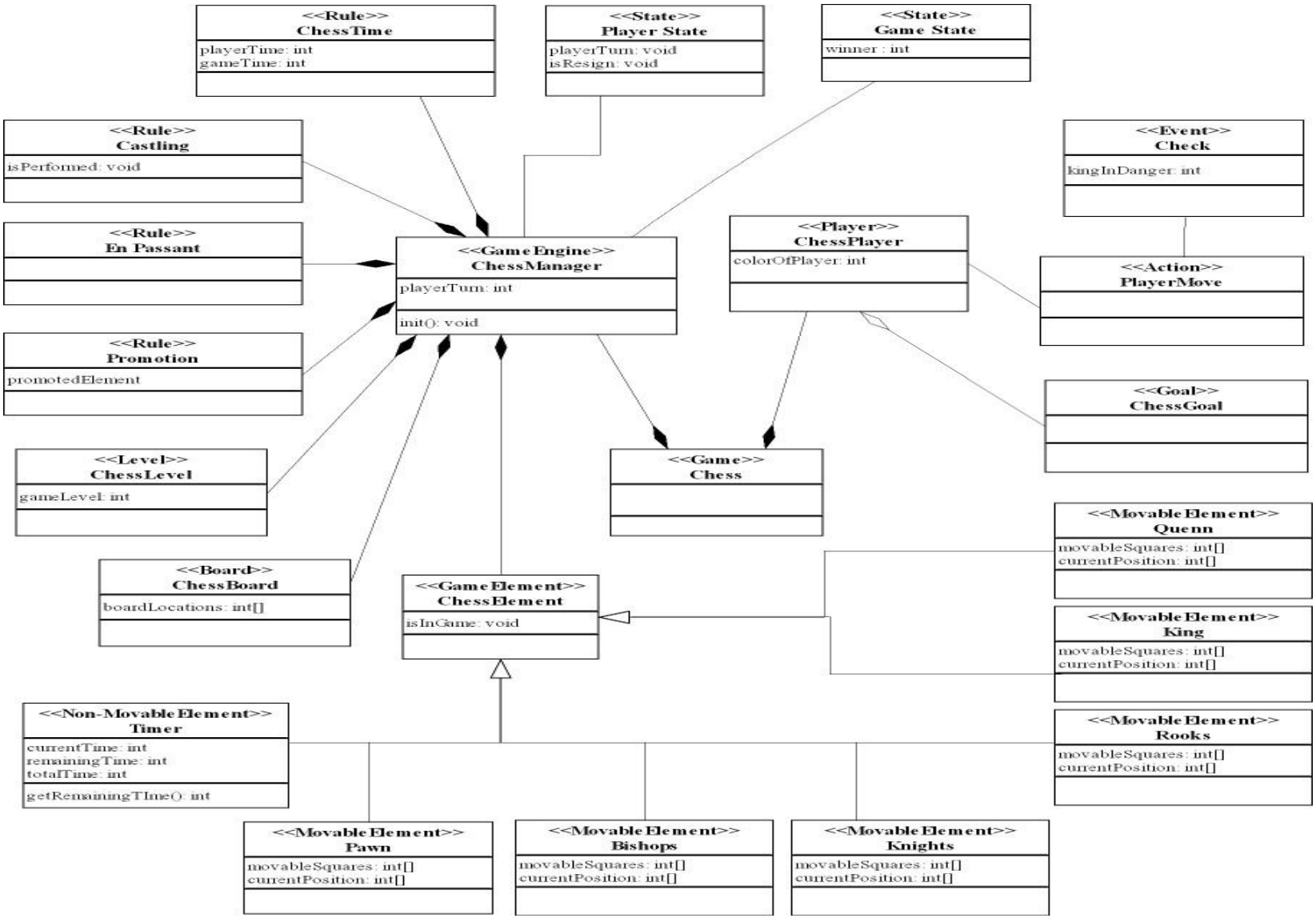
Metaclasses for UML Profile for Board Games

Game Model Element	Stereotype	UML Metaclass
Game	BGGame	Class
GameEngine	BGEngine	Component
GameElement	BGElement	Class
Player	BGPlayer	Class
Event	BGEvent	Class
Action	BGAction	Class
GameState	BGState	State
Board	BGBoard	Class
Goal	BGGoal	Class
MovableElement	BGMovableElement	Class
NonmovableElement	BGNonmovableElement	Class
Rule	BGRule	Class
Level	BGLevel	Class

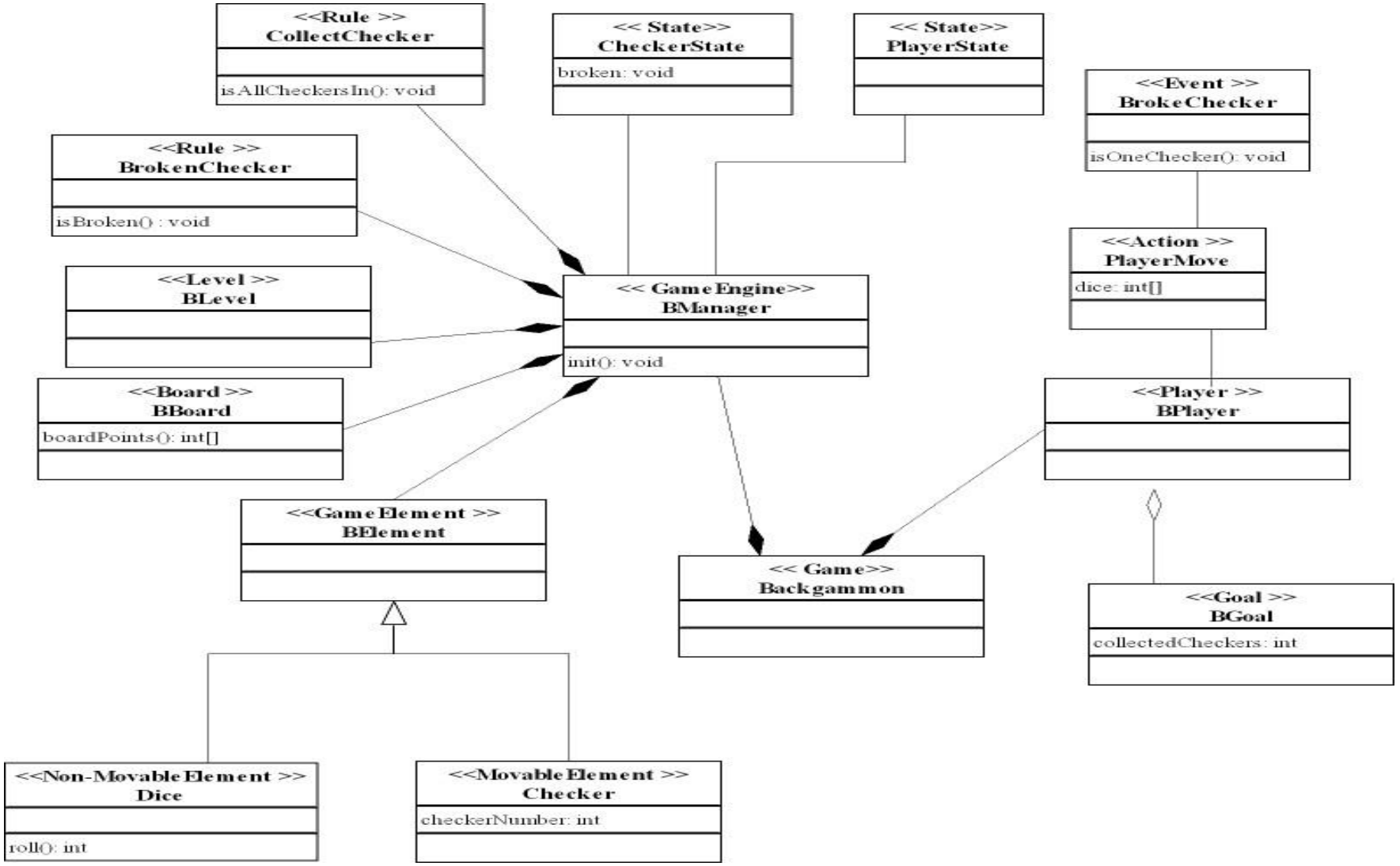
UML Profile


<<Profile>> BoardGames
<<enumeration>>
PlayerType
Human
Computer
<<enumeration>>
Coordinate
1D
2D
<<stereotype>>
Player
-type: PlayerType
<<stereotype>>
Level
<<stereotype>>
Rule
<<Metaclass>>
Class
<<stereotype>>
GameElement
<<stereotype>>
Event
<<stereotype>>
Action
<<stereotype>>
Goal
<<stereotype>>
MovableElement
-coordSpace: Coordinate
<<stereotype>>
NonMovableElement
<<Metaclass>>
State
<<stereotype>>
Board
<<Metaclass>>
Component
<<stereotype>>
GameEngine
<<stereotype>>
GameState

Sample Model - Chess



Sample Model - Backgammon



Model to Text Transformations

- The Model to Text transformation addresses how to translate a model to various text artifacts such as code, deployment specifications, reports, documents, etc.
- Essentially, the m2t standard needs to address how to transform a model into a text representation.

Model to Text Transformation

- In this project, we derived model to text transformation by using openarchitectureware tool.
- We created an initial structuring file for the chess game.

OPENARCHITECTUREWARE GAME METAMODEL XSD FILE

```
...
 <complexType name="Game">
  <sequence>
    <element name="start" type="IDREF"/>
    <element name="gameEngine" type="tns:GameEngine"/>
  </sequence>
 </complexType>
  <complexType name="GameEngine">
  <sequence>
    <element name="gameName" type="string"/>
    <element name="gameElement" type="tns:GameElement"/>
    <element name="nonmovableElement" type="tns:NonmovableElement"/>
    ...
   </sequence>
 </complexType>
    <complexType name="Player">
  <sequence>
    <element name="playerName" type="string"/>
    <element name="action" type="tns:Action"/>
</sequence>
 </complexType>
...
/schema>
```

OPENARCHITECTUREWARE CHESS GAME MODEL XML FILE

```
...
<start>Chess Game Metamodel Structure</start>
 <gameEngine>
  <gameName>Chess Game</gameName>
  <gameElement>
    <gameElementType>Movable Elements, Nonmovable
   Elements</gameElementType>
  </gameElement>
...
  <movableElement>
    <name>king, queen, rooks, bishops, knights, pawns</name>
  </movableElement>
  <board>
    <xCoordinates>a, b, c, d, e, f, g, h</xCoordinates>
    <yCoordinates>1, 2, 3, 4, 5, 6, 7, 8</yCoordinates>
  </board>
  <player>
    <playerName>Player Name</playerName>
    <action>
      <coordinateX>Player's move x-coordinate</coordinateX>
      ...
  </player>
 </gameEngine>
</game>
```

OPENARCHITECTUREWARE TRANSFORMATION TEMPLATE FILE

«IMPORT metamodel»

«DEFINE Root FOR metamodel::Game»
«FILE "ChessGame.game"»
Explanation: «start»
Game Name: «gameEngine.gameName»
Game Element Types: «gameElement.gameElementType»
Non-movable Element Types: «nonmovableElement.name»
Movable Element Types: «movableElement.name»
Board X-Coordinates: «board.xCoordinates»
Board Y-Coordinates: «board.yCoordinates»
Player Name: «player.playerName»
Player's Action X-Coordinate: «player.action.coordinateX»
Player's Action Y-Coordinate: «player.action.coordinateY»
«ENDFILE»
«ENDDEFINE»

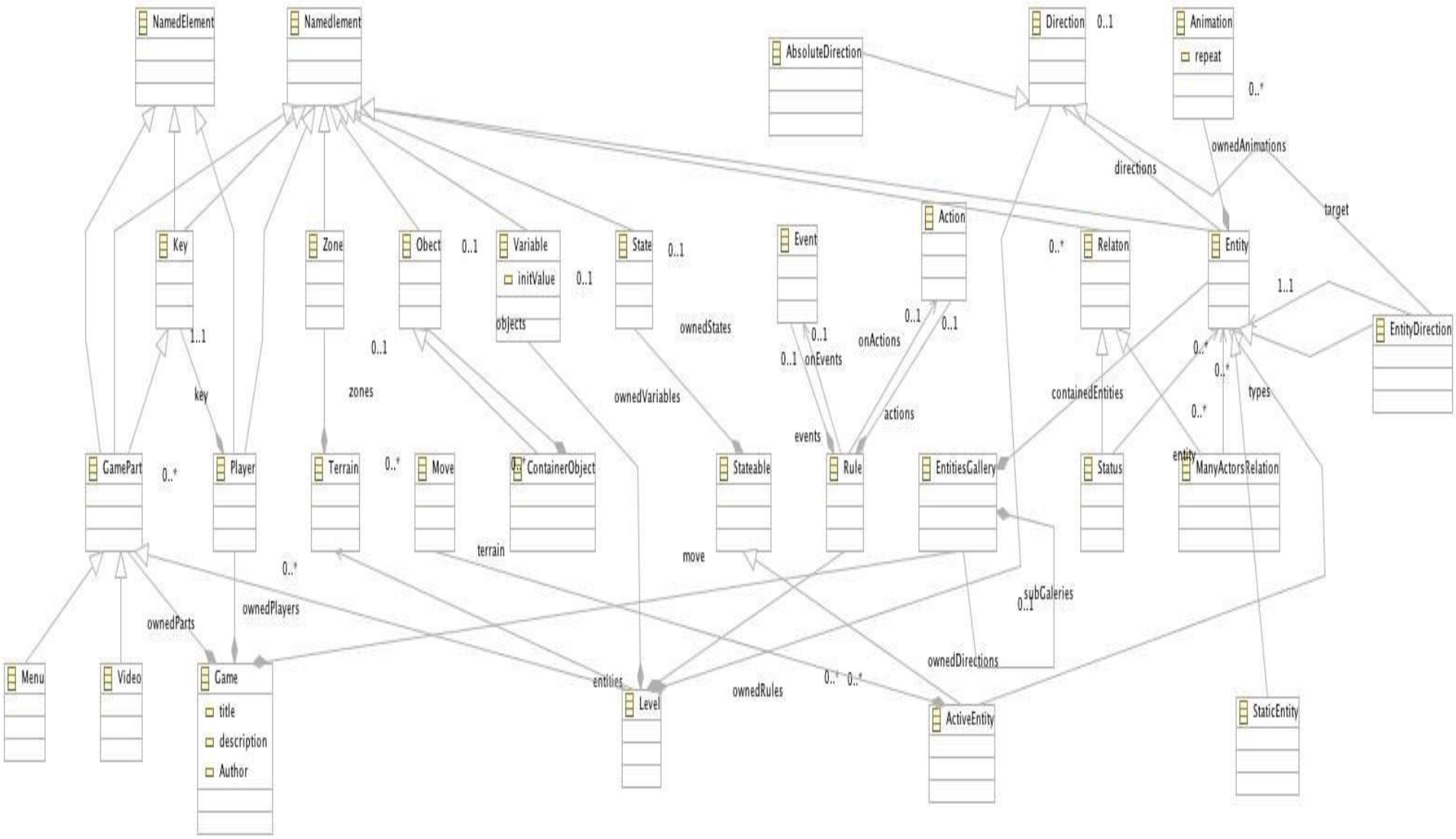
Model to Model Transformations

- Interoperability is one of the key issues of MDSD
  - Tool interoperability, ability to port models between tools that conform to different metamodels.
  - M2M transformations allows us to transform a source model to a desired target model.
  - Then we are able to use out model within other modeling tool which accepst target metamodel

M2M Transformations (cont…)

- Our domain model describes a sub domain of games, Board Games
- We aim to be able to transform our models to a more general domain model, Game DSL
- ATL – Atlas Transformation Language is used for M2M transformations.

Game DSL


NamedElement
Namedlement
AbsoluteDirection
Direction
0..1
Animation
repeat
0..*
ownedAnimations
directions
target
Key
Zone
Obect
0..1
Variable
initValue
0..1
objects
State
0..1
ownedStates
Event
Action
0..*
Relaton
Entity
1..1
EntityDirection
1..1
key
0..1
zones
ownedVariables
0..1
onEvents
onActions
0..1
0..1
actions
events
containedEntities
0..*
0..*
types
0..*
entity
GamePart
0..*
Player
Terrain
0..*
Move
0..*
ContainerObject
Stateable
Rule
EntitiesGallery
Status
ManyActorsRelation
terrain
move
0..*
subGaleries
0..1
ownedParts
ownedPlayers
ownedDirections
Menu
Video
Game
title
description
Author
entities
Level
ownedRules
0..*
0..*
ActiveEntity
StaticEntity

Mapping

- M2M transformation is basically the process of defining a mapping between a source and target metamodel.
- Once the mapping is defined, it can be used for transforming instance models which conform to source metamodel to corresponding models which conforms to target metamodel.

Mappings

- GameEngine <-> Game
- MoveableElement <-> ActiveEntity
- NonMoveableElement <-> StaticEntity
- Board <-> ContainerObject
- Level <-> Level
- Action <-> Action
- State <-> State
- Event <-> Event

ATL Mappings

- ATL mappings are performed by rules:

```
rule GameEngine2Game{
    from
        ge: BoardGame!GameEngine
    to
        g: GameDSL!Game(
                title <- ‘ChessGame',
                Author <- 'DEG',
                description <- 'Generated sample chess game'
        )
}
```

ATL Mappings

```
rule MovableElement2ActiveEntity{
    from
        me: BoardGame!MovableElement
    to
        ae: GameDSL!ActiveEntity(
            name <- me.name
        )
}

rule NonMovableElement2StaticEntity{
    from
        nme: BoardGame!NonMovableElement
    to
        ne: GameDSL!StaticEntity(
            name <- nme.name
        )
}
```

Conclusion

- Using MDSD approach have such advantages;
  - Automated transformations and DSL enhance **software quality**
  - Defined architectures, modeling languages can be used in the sense of software product line which leads to higher level of **reusability**.
  - Through higher level of abstraction, **manageability of complexity** improved.
  - More productive environment
  - With standardization, **interoperability** and **portability** of software systems are improved.
  - MDSD allows an abstraction between implementation and the solution which brings crutial advantages to companies in software development process.
- Current Difficulties
  - Since the tools for MDSD are not mature yet , we have difficulties in the entire process using MDSD tools.

Questions