Theory and Practice of Game Object Component Architecture

Marcin Chady

Radical Entertainment

GDC Canada

www.GDC-Canada.com

GDC Canada

www.GDC-Canada.com

THINK SERVICES

REBOOT

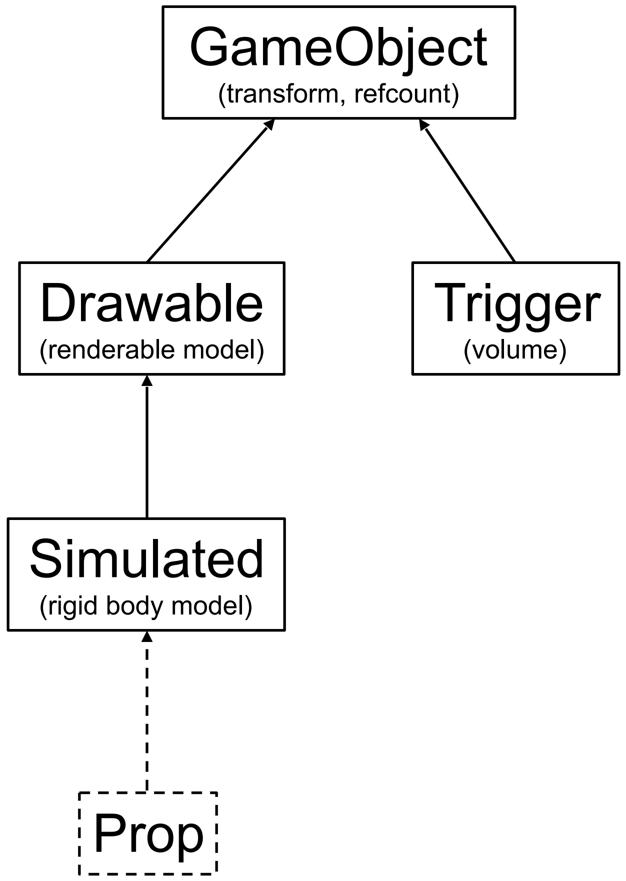
Outline

- Component-Oriented vs Object-Oriented Programming
- Radical’s approach
- Results from [PROTOTYPE]

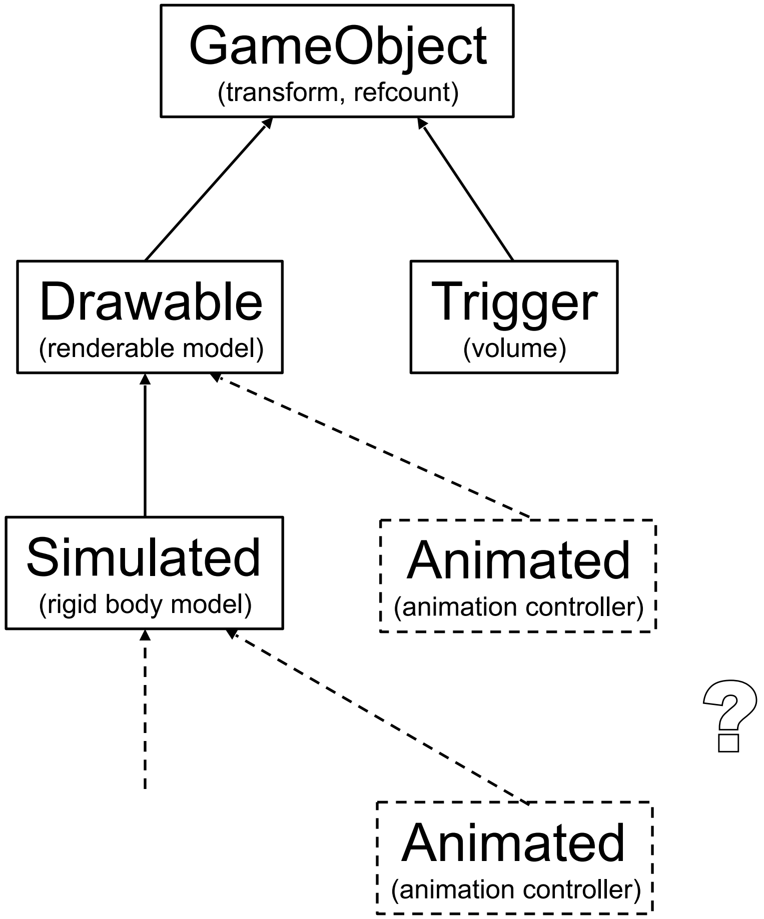
What are Game Objects?

- Anything that has a representation in the game world
  - Characters, props, vehicles, missiles, cameras, trigger volumes, lights, etc.
- Need for a standard ontology
  - Clarity
  - Uniformity
  - Feature, staff and tool mobility
  - Code reuse
  - Maintenance
    - E.g. use of modularity/inheritance reduces duplication

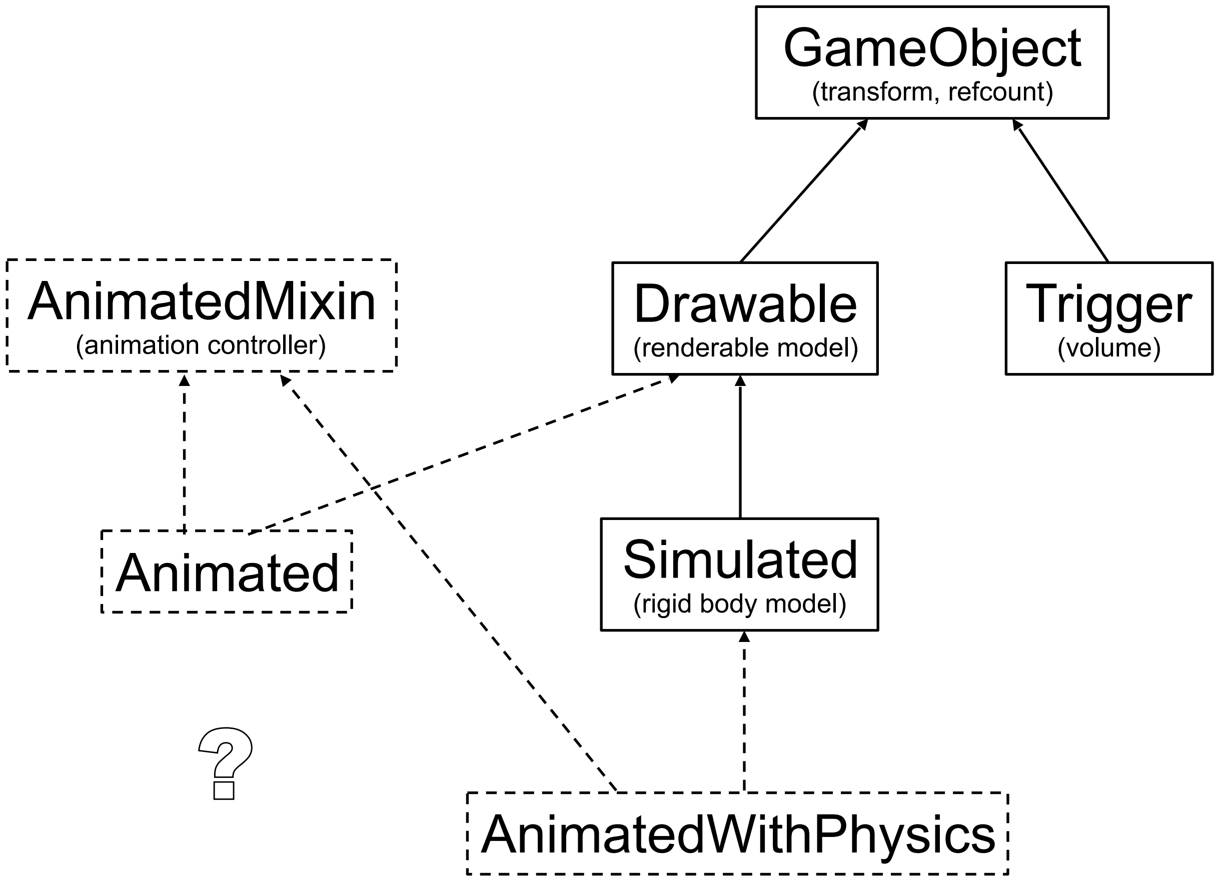
A Game Object Class Hierarchy



Adding Stuff



Mix-ins Perhaps?



Observations

- Not every set of relationships can be described in a directed acyclic graph
- Class hierarchies are hard to change
- Functionality drifts upwards
- Specialisations pay the memory cost of the functionality in siblings and cousins

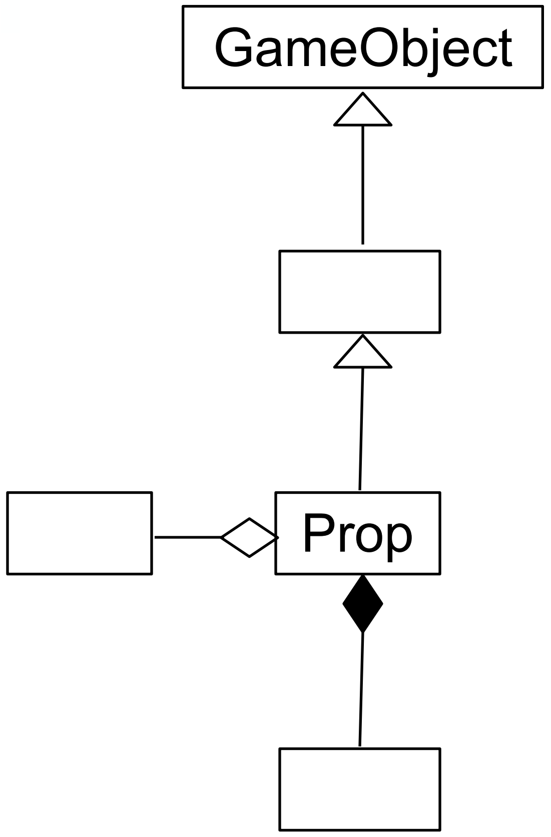
Change

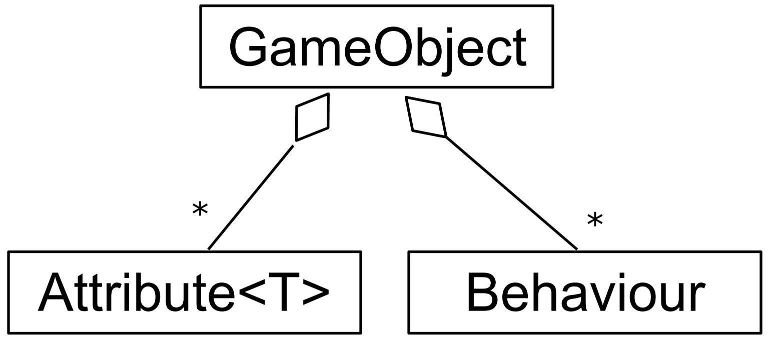
- You can ignore it
- You can resist it
- Or you can embrace it
- But you cannot stop it

Component-Based Approach

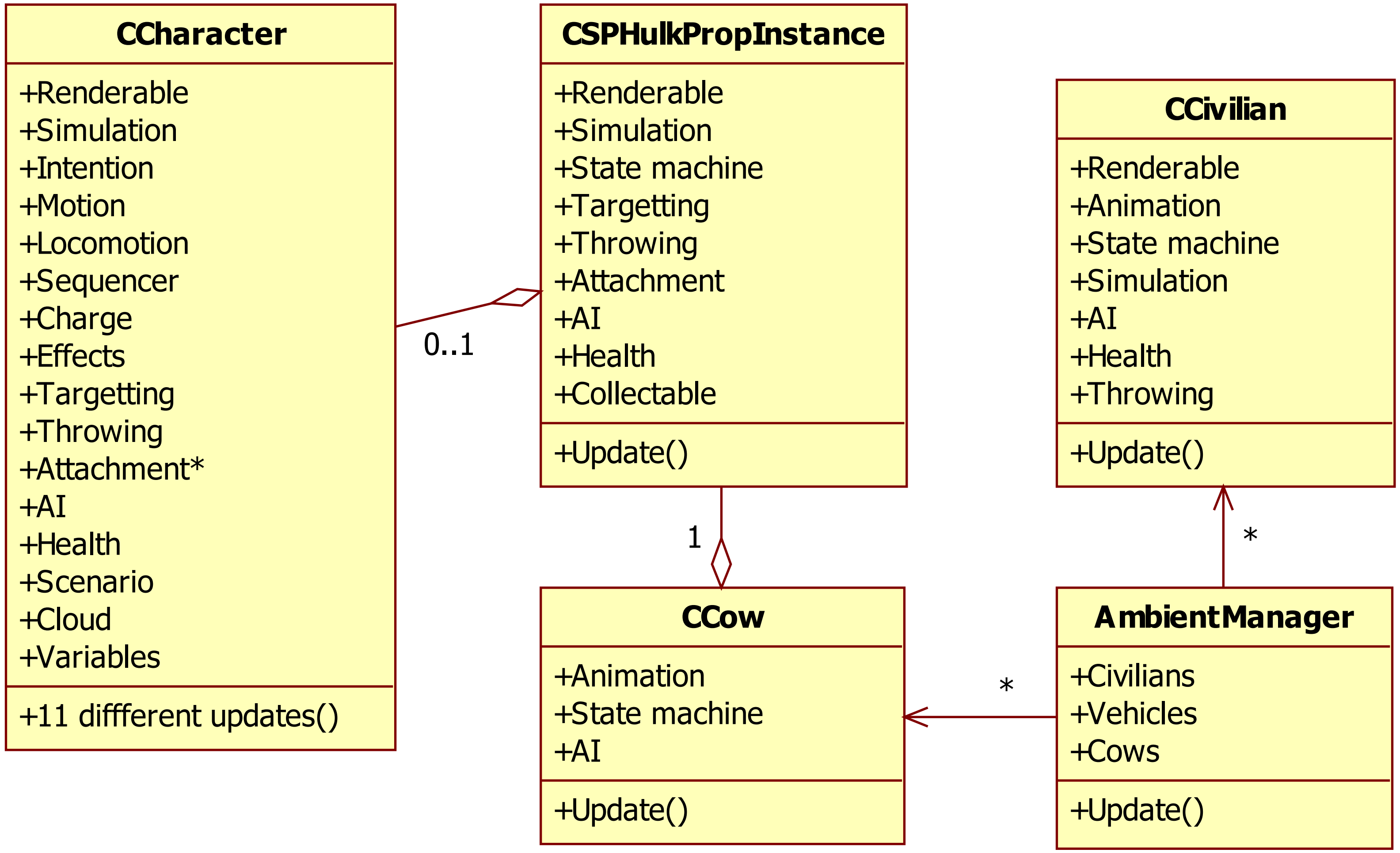
- Related to, but not the same as aspect-oriented programming
- One class, a **container** for:
  - attributes (data)
  - behaviours (logic)
- Attributes := list of key-value pairs
- Behaviour := object with OnUpdate() and OnMessage()

Components vs Hierarchies





Hulk:UD Object Model



Prototype Game Objects

Alex	Helicopter	Pedestrian(HLOD)	Pedestrian(LLOD)
PhysicsBehaviour	PhysicsBehaviour	PhysicsBehaviour	
TouchBehaviour	TouchBehaviour		
CharacterIntentionBehaviour	CharacterIntentionBehaviour	CharacterIntentionBehaviour	
MotionTreeBehaviour	MotionTreeBehaviour	MotionTreeBehaviour	
CollisionActionBehaviour	CollisionActionBehaviour		
PuppetBehaviour	PuppetBehaviour	PuppetBehaviour	
CharacterMotionBehaviour			
MotionStateBehaviour			
RagdollBehaviour			
CharacterSolverBehaviour	CharacterSolverBehaviour		
HealthBehaviour	HealthBehaviour	HealthBehaviour	
RenderBehaviour	RenderBehaviour	RenderBehaviour	
SensesInfoBehaviour			SensesInfoBehaviour
HitReactionBehaviour	HitReactionBehaviour		
GrabSelectionBehaviour			
GrabbableBehaviour	GrabbableBehaviour	GrabbableBehaviour	
	GrabBehavior	GrabBehaviour	
TargetableBehaviour	TargetableBehaviour	TargetableBehaviour	TargetableBehaviour
AudioEmitterBehaviour	AudioEmitterBehaviour	AudioEmitterBehaviour	
FightVariablesBehaviour	FightVariablesBehaviour		
	EmotionalStateBehaviour	EmotionalStateBehaviour	
ThreatReceiverBehaviour	ThreatReceiverBehaviour		
	FEDisplayBehaviour	FEDisplayBehaviour	
		CharacterPedBehaviour	PedBehaviour

Data-Driven Creation

```
TOD_BeginObject GameObject 1 "hotdog_concession"
{
    behaviours
    {
        PhysicsBehaviour 1
        {
            physicsObject "hotdog_concession"
        } ,
        RenderBehaviour 1
        {
            drawableSource  "hotdog_concession"
        } ,
        HealthBehaviour 1
        {
            health 2.000000
        } ,
        GrabbableBehaviour 1
        {
            grabbableClass "2hnd"
        }
    }
}
TOD_EndObject
```

- Text or binary
- Loaded from pipeline
- Load and go
- Delayed instancing
- Dedicated tools
- Data-driven inheritance

Advantages

- Endowing with new properties is easy
- Creating new types of entities is easy
- Behaviours are portable and reusable
- Code that talks to game objects is type-agnostic
- Everything is packaged and designed to talk to each other
- In short: you can write generic code

Disadvantages

- In short: you have to write generic code
- Game objects are typeless and opaque
- Can’t ask, e.g.

```
if object has AttachableBehaviour
   then attach to it
```

This is wrong!

- Code has to treat all objects identically

Messaging

```
AttachMessage msg(this);
object->OnMessage(&msg);
```

- Dispatched immediately to all interested behaviours (synchronous operation)
- Fast, but not as fast as a function call
- Use for irregular (unscheduled) processing
  - Collisions, state transitions, event handling
- Can be used for returning values

Attribute Access

- The game object must be notified if you modify an attribute
- Const accessor
  - Read-only access
  - Cacheable
- Non-const accessor
  - Permits writing
  - Not cacheable
  - Sends a notification message to the game object
- Free access from object’s own behaviours

An attribute or not an attribute?

- Attribute if
  - accessed by more than one behaviour, or
  - accessed by external code
- Otherwise a private member of the behaviour
- If not sure, make it an attribute

Game Object Update

-
  ```
  GameObject::OnUpdate(pass, delta)
      for b in behaviours
              b.OnUpdate(pass, delta)
  ```
- `OnUpdate()` and `OnMessage()` are the only two entry points to a behaviour.

HealthBehaviour Example

```
void HealthBehaviour::OnMessage(Message* m)
{
    switch (m.type)
    {
        case APPLY_DAMAGE:
            Attribute<float>* healthAttr = GetAttribute(HEALTH_KEY);
            healthAttr->value -= m.damage;
            if (healthAttr->value < 0.f)
                mGameObject->SetLogicState(DEAD);
            break;

        case ATTR_UPDATED:
            if (m.key == HEALTH_KEY)
            {
                Attribute<float>* healthAttr = GetAttribute(HEALTH_KEY);
                if (healthAttr->value < 0.f)
                    mGameObject->SetLogicState(DEAD);
            }
            break;
    }
}
```

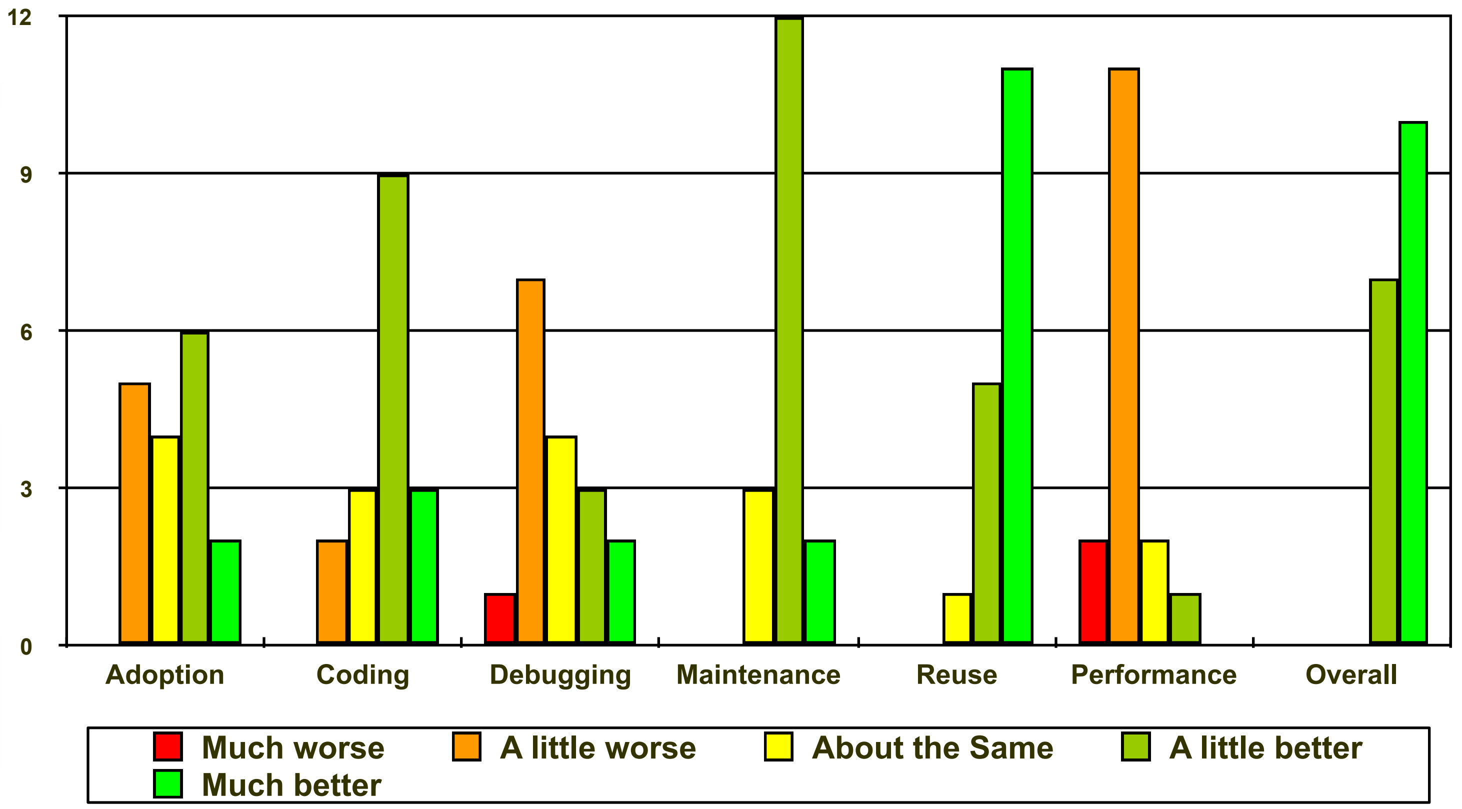
Components in Practice

Behaviours and Attributes
in [PROTOTYPE]

Adoption

- Some coders were resistant:
  - Too complicated
    - Don’t know what’s going on
  - Too cumbersome
    - Calling a function is easier than sending a message
    - Reading a data member is easier than retrieving an attribute
  - Don’t like typeless objects
- Ongoing education

Post-Mortem Survey



Post-Mortem Comments

- Data-driven creation was the biggest win
- Prototyping is the biggest win once you have a library of behaviours
- Modularity of behaviours was the biggest win
- Data inheritance was the biggest win
- Components are nothing new - no modern game could be built without them

Performance

- GameObject::OnUpdate and OnMessage are easy targets
  - For the critic
  - For the optimiser
- Existing optimisations:
  - Message masks
  - Update masks
  - Logic state masks
  - Time-slicing
  - Attribute caching
  - Leaving the back door open

Performance Lessons

- Best optimisations are algorithmic:
  - Avoid unnecessary messages, e.g.

```
object->OnMessage(&message1);
if (message1.x)
        object->OnMessage(&message2);
```

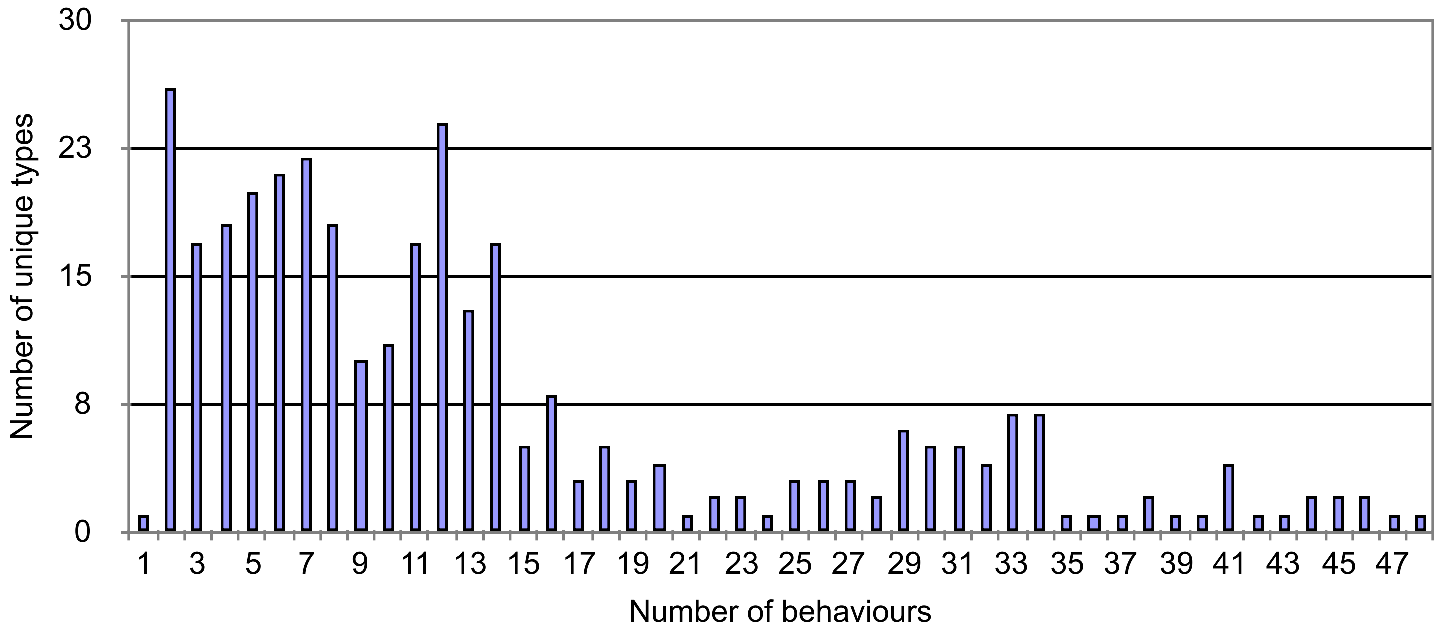
  - Prefer attributes over messages
  - Avoid unnecessary updates
- Better instrumentation
- Legalise the back door entrance

Future Improvements

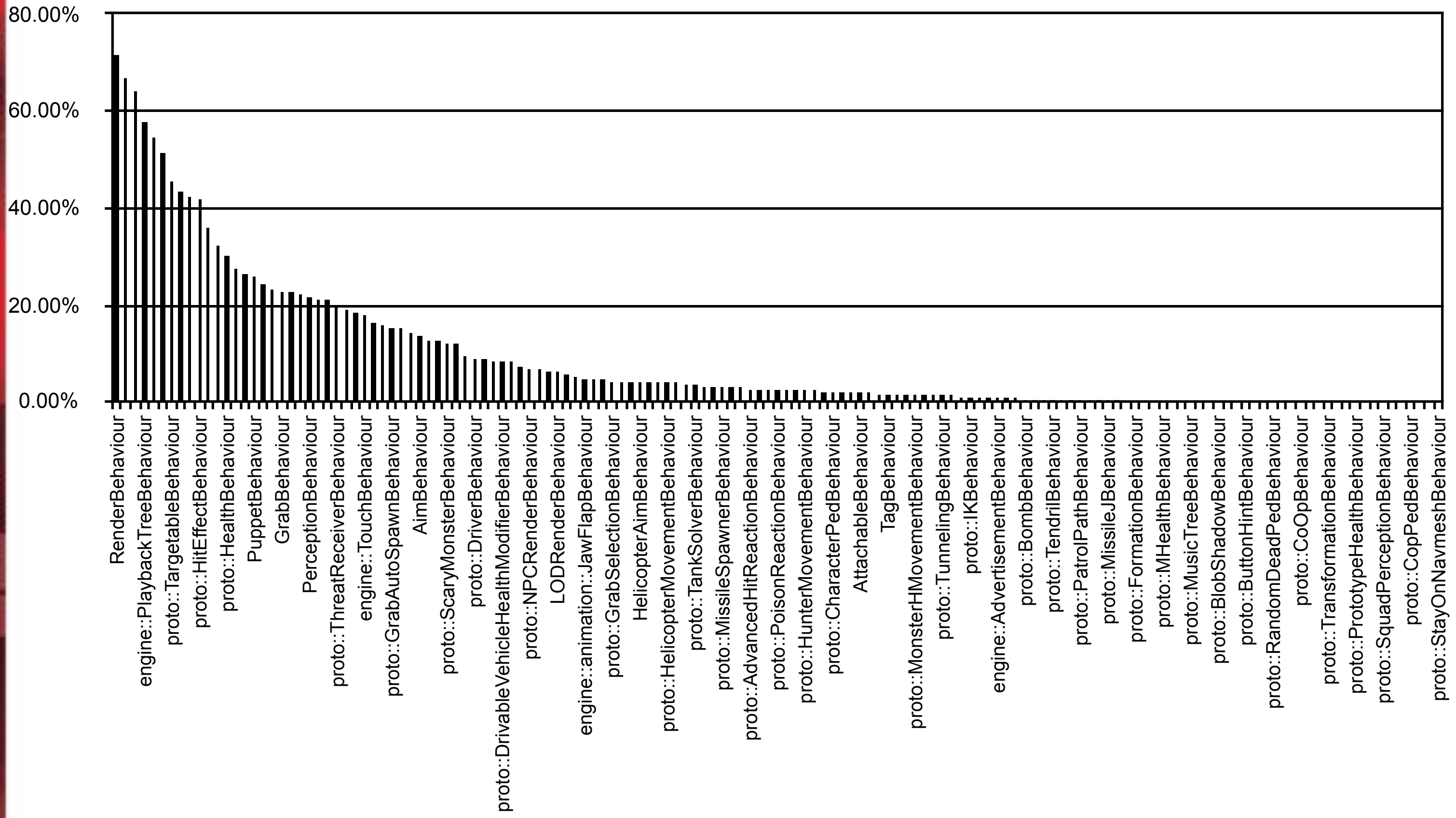
- Stateless behaviours
- Submit batches of objects to stateless behaviours
  - Better suited for parallel architectures
- Message queuing

Prototype's Data Types

- 1544 game object definitions
- 145 unique behaviours
- 335 unique data types
  - 156 unique prop types alone

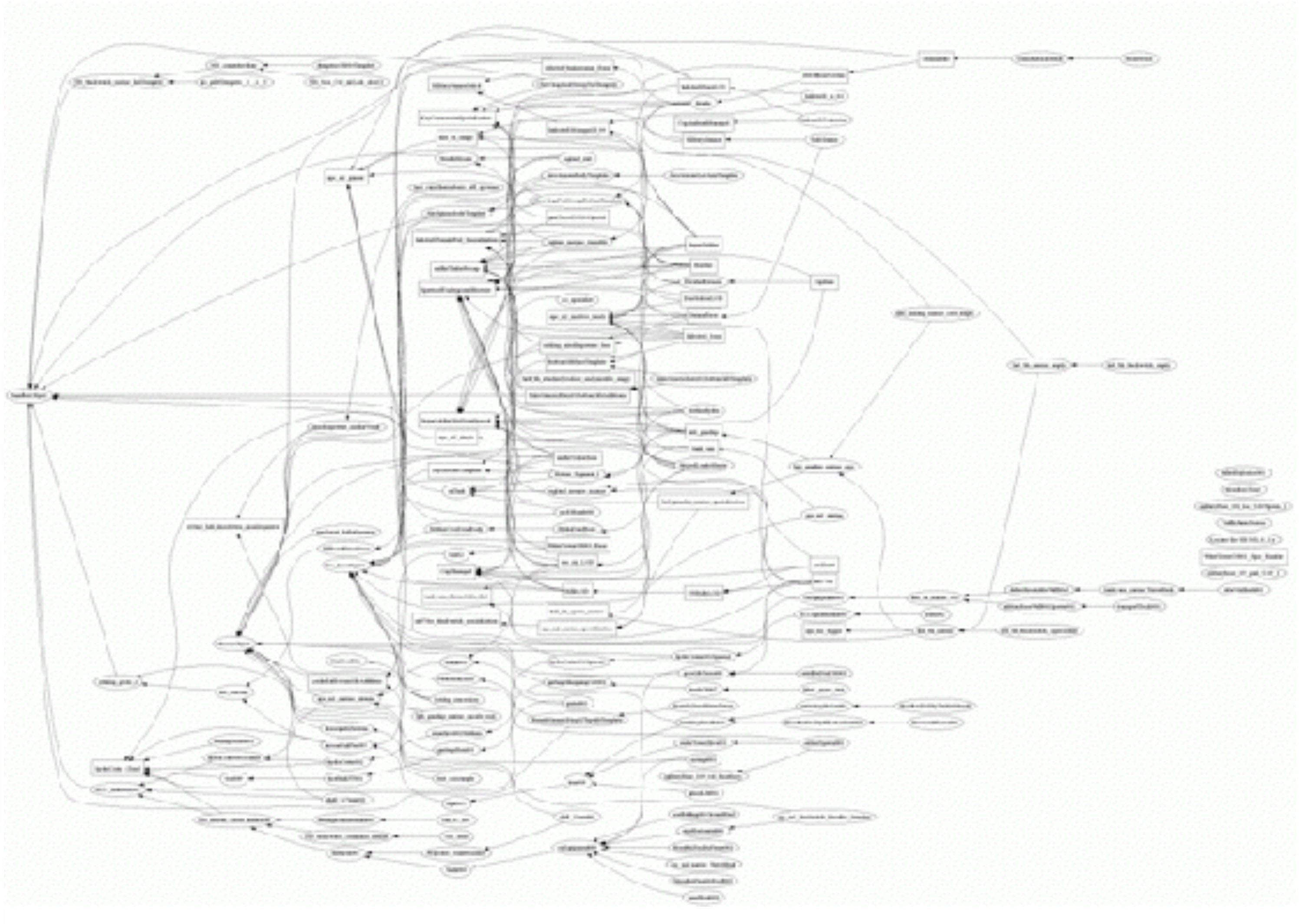


Behaviour Usage



Implicit “Class Hierarchy”

Implicit "Class Hierarchy"



GDC Canada

www.GDC-Canada.com

Summary

- Designs change
- Class hierarchies don’t like change
- Components do, but not without some sacrifices