Predicates RFC

Metacosm Development Team

Revision History

Revision M1July, 15th 2001
First version

This document is meant to replace Conditional Expressions RFC by refining and extending the initial concepts.

1. License

2. Metacosm

For more information on Metacosm, check our site: Metacosm (http://metacosm.org).

3. Goals

The goal of this RFC is to define the concepts needed to the implementation of a generic and extensible Predicate system. This predicate system is meant to be used extensively by Metacosm's interaction engine, in particular for dynamic Actions.

We present a possible solution from an implementation point of view.

4. Requirements

In the context of the interaction engine, we need to be able to define conditions guarding modifications on Entities. More specifically, we would like to be able to prevent the execution of an Action if the performing Entity is not strong enough, as an example.

We'd also like to be able to define conditions that direct the execution path of Actions. If we consider those as state machines, we want to be able to guard transitions depending on conditions such as the result of a precedent Action.

5. Definitions

Predicate

Something that is affirmed or denied of the subject in a proposition in logic. (Merriam-Webster's Collegiate Dictionary)

In Metacosm: An object evaluating a predicate on a set of Entities before to determine if manipulation on them are allowed. If an Entity manipulation (Action) is guarded by a Predicate, it can only be executed if and only if the guarding Predicate is validated (i.e., it is evaluated as true). A Predicate defines conditions to be verified on a set of Entities. Predicates are implemented as Command (design pattern) objects. As such, they can be serialized and reused in lots of different contexts without needed to be described again.

Attribute (for a Predicate)

An inherent characteristic. (Merriam-Webster's Collegiate Dictionary)

In Metacosm: An attribute is a characteristic associated to a *Predicate* against which the particular aspect of the *Predicable* tested by this Predicate, is evaluated.

Predicable

Something that may be predicated. (Merriam-Webster's Collegiate Dictionary)

In Metacosm: An object on which *Predicates* can operate.

Operator

In Metacosm: An object able to manipulate *Predicates*. In particular, an object able to combine several Predicates into one.

6. Description

A Predicate is defined as:

- a mnemonic name
- a human-readable description
- a formal definition

A Predicate is identified by its *mnemonic name* which is used to refer to a given Predicate. A Predicate's mnemonic is unique throughout a given Metacosm instance. Mnemonics are user- and editor-friendly since they describe the Predicate in a very concise way.

The *human-readable description* expands on the mnemonic and precise what the Predicate is about.

The *formal definition* specifies in an unambiguous way (i.e. using a well defined grammar) what condition the Predicate checks. See the Predicate grammar section for more details.

7. Categories of Predicates

Next, we categorize the different types of Predicates that we have identified.

7.1. Special Predicates

7.1.1. True Predicate

This Predicate is always evaluated to true.

7.1.2. False Predicate

This Predicate is always evaluated to false.

7.2. Unary Predicates

7.2.1. Is Predicate

evaluated to true if the current Predicable is an Entity and has the same identity has the one defined by the Predicate’s attribute.

7.2.2. InfluencedBy Predicate

This Predicate is evaluated to true if the current Predicable is influenced by the Influence identified by the Predicate’s attribute.

7.2.3. Possess Predicate

This Predicate is evaluated to true if the Predicable is in possession of the Entity identified by the Predicate’s attribute.

7.2.4. CanPerform Predicate

This Predicate is evaluated to true if the Predicable can perform the Action identified by the Predicate’s attribute.

7.3. Binary Predicates

Binary Predicates are more complex Predicates which attributes are composed by a couple of values, named first and second argument respectively. Next, we detail binary Predicates.

7.3.1. LesserThan Predicate

First argument: Property name
Second argument: Threshold value

This Predicate is evaluated to true if the current Predicable has a Property named like the first argument and this Property has a value that is lesser than the value given by the second argument.

7.3.2. GreaterThan Predicate

First argument: Property name
Second argument: Threshold value

This Predicate is evaluated to true if the current Predicable has a Property named like the first argument and this Property has a value that is greater than the value given by the second argument.

7.3.3. EqualsTo Predicate

First argument: Property name
Second argument: Threshold value

This Predicate is evaluated to true if the current Predicable has a Property named like the first argument and this Property has a value that is equals to the value given by the second argument.

7.3.4. KnowsAbout Predicate

First argument: Entity identifier
Second argument: Level of knowledge

This Predicate is evaluated to true if the current Predicable knows about the Entity identified by the first argument with a level given by the second argument.

7.3.5. RelatesTo Predicate

First argument: Entity identifier
Second argument: Type of relation

This Predicate is evaluated to true if the current Predicable is related to the Entity identified by the first argument by a link given by the second argument.

8. Predicate grammar

We define a Predicate grammar that describes how Predicates can be combined to form more elaborate Predicates.

```
predicate -> orPredicate
orPredicate -> andPredicate or
            orPredicate
orPredicate -> andPredicate
andPredicate -> simplePredicate and
             andPredicate
andPredicate -> simplePredicate
simplePredicate -> ( orPredicate )
simplePredicate -> notPredicate
notPredicate -> not simplePredicate
```

Terminals are displayed like so: *terminal*

Note that this context-free grammar defines a precedence on rules. In the preceeding grammar, *and* has a higher precedence than *or*.

Using this grammar, it is possible to define fairly complex combinations of Predicates that can describe complex pre-conditions. This set of Predicates is evaluated to a single Predicate according to the grammar rules.

9. Examples

In the following, examples are title after their *description* (see Description section).

Example 1. Guards with a sword

Mnemonic: guard-sword

Definition: (InfluencedBy Guard) and (Possess Sword)

Example 2. Non-elven guards with a axe, knowing Raoul the baker

Mnemonic: guard-axe
Definition: (InfluencedBy Guard) and not (InfluencedBy Elf) and (Possess Axe) and (KnowsAbout "Raoul the baker" a_little)

Example 3. Able to bribe a sergeant

Mnemonic: able-bribe-sergeant
Definition: (CanPerform bribe) and (GreatherThan bribe_skill 50%)

10. Implementation

The generic form a *Predicate* is materialized by a generic interface that allows Predicates to be evaluated. Since Predicates can be used to guard the execution of Actions, their evaluation must take at least the same parameters defined by the Action interface. We thus define the following interface:

```
public interface Predicate {
  boolean evaluate(Entity[] sources,
  Entity[] targets,
  Object[] parameters);
}
```

This interface is associated with *Predicable* objects. This interface defines the methods needed by *Predicable* and *Predicate* objects to communicate efficiently. Obvious

Predicables are Actions but they are not restricted to these. We can also imagine that spells, Quests, IAs or other concepts could be implemented as Predicables.

```
(to be defined)
```