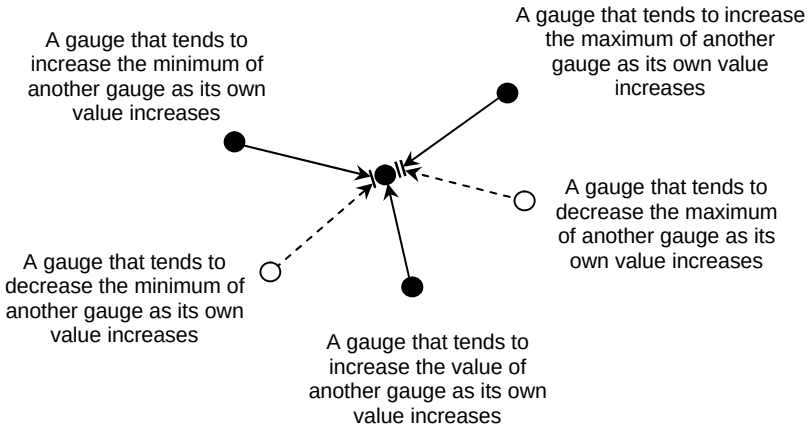
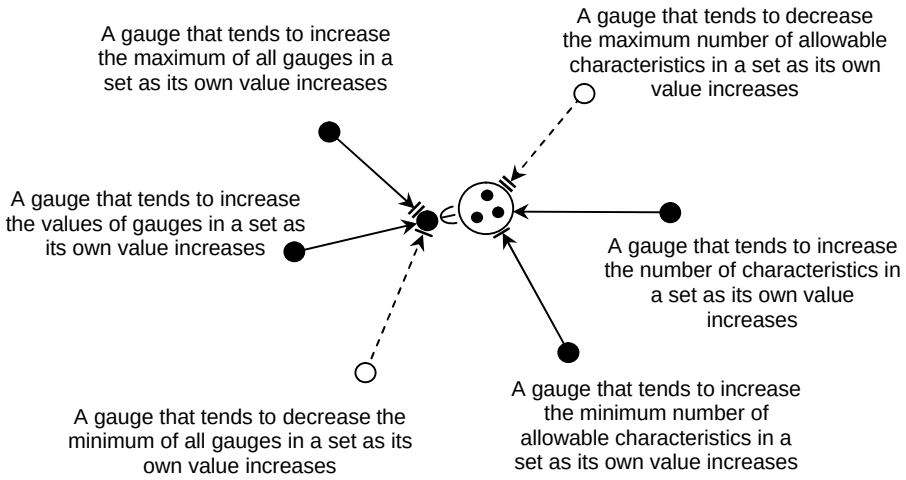
Proposed RPG Diagram Notation Changes

Revision 5

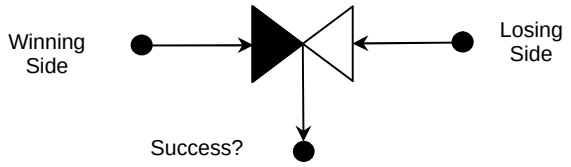
Gauge Diagrams



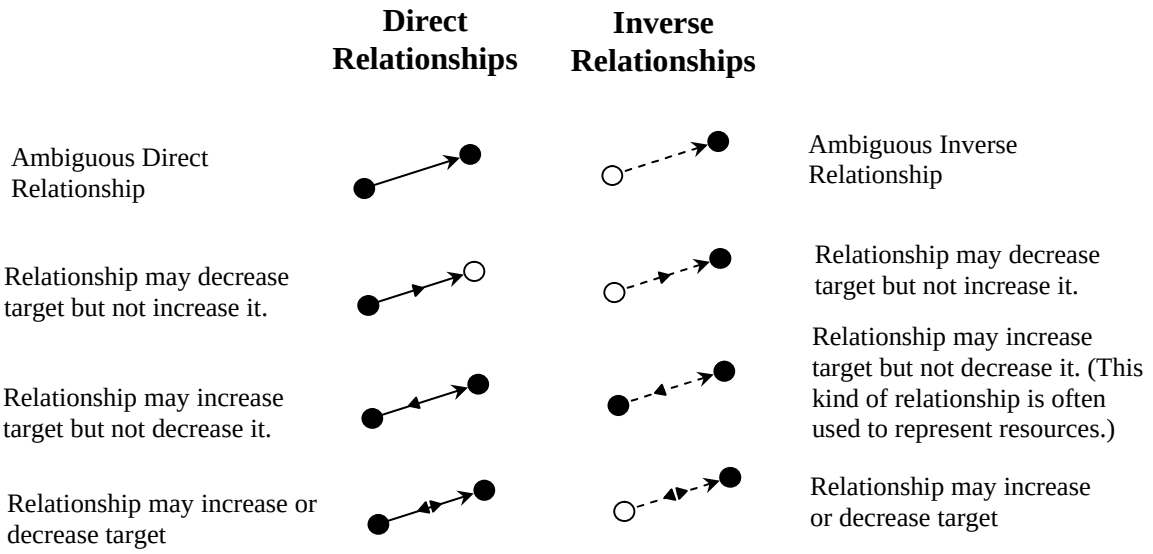
An "Element of" symbol: ∈ can be used to identify a gauge as being an element of a set. In this way, relationships can specify whether they apply to individual set elements or to the set as a whole:



If you want to distinguish the winning side from the losing side in a contest, fill in the winning side of the contest icon.



It would sometimes be useful to more clearly specify the precise nature of a relationship. Some relationships can increase, but not decrease the target. Some can decrease but not increase the target. Others can do both. To allow more detail, adornments can be added to provide this information:

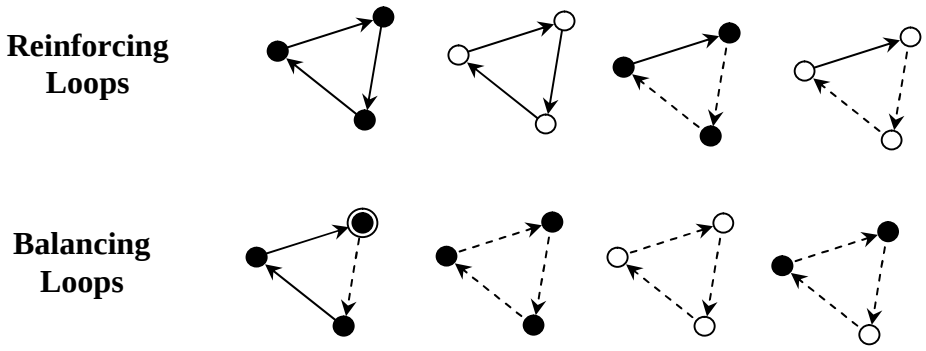


The triangles are placed on the lines so that you can read diagrams easily. Just remember these simple rules:

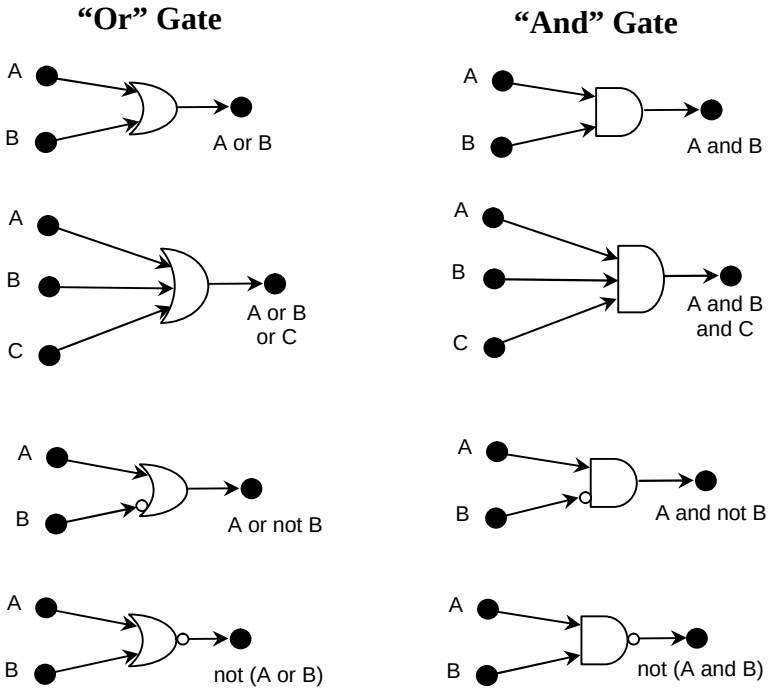
1) If the base (big end) of a triangle points toward the target, the relationship increases the target.
2) If the tip (little end) of a triangle points toward the target, the relationship decreases the target.

Note that this changes the definition slightly from the book's definition of what a solid arrow and dashed arrow mean. In the book, a solid arrow means that an increase in the originating gauge implies an increase in the target gauge. A dashed arrow means that an increase in the originating gauge implies a decrease in the target gauge. Nothing more is specified. Now, a solid arrow means that the originating and target gauges change in the same direction (up or down) while a dashed arrow means that the originating and target gauges change in the opposite direction. This redefinition does not actually change the diagrams in the book (at least, I don't think so), but allows for more clarity when needed.

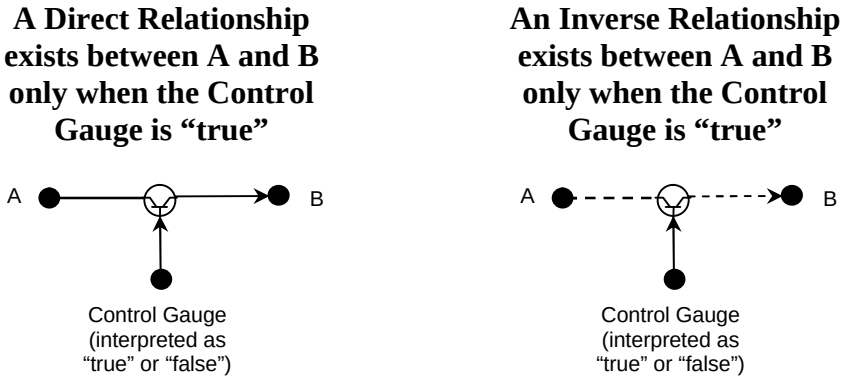
Loops in Gauge Diagrams highlight a game's reward systems and balancing mechanisms. Reinforcing Loops have an even number of inverse relationships. Balancing Loops have an odd number of inverse relationships. This isn't an actual change to the technique. It is only an illustration of the concept.



Sometimes, currency flow depends on a logical operation, such as “The attribute value goes up if the roll succeeds and the player spends a token”. To model these kinds of operations, we borrow the “and”, “or”, and “not” gates from digital circuit design. The output from these gates is the logical “or” or “and” of the inputs. The “not” gate, represented by a small circle, transforms a “true” to a “false” or a “false” to a “true”. In gauge diagrams, “false” is any value of 0 while “true” is any non-zero value. If you need to assign a numerical value to the output of a gate, assume “true” is 1 and “false” is 0.

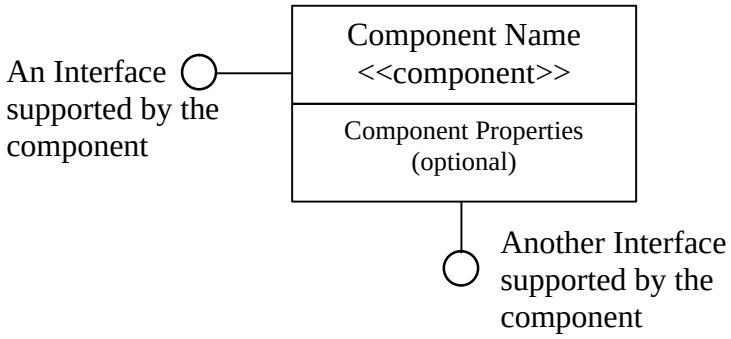


At other times, it is useful to be able to specify that currency flow only occurs upon certain conditions. For example, “If your attribute value falls to zero, the player with the most tokens must give you one token.” To represent these kinds of currency flows, we borrow the transistor icon from electric circuit diagrams. Our idealized transistor acts as a sort of switch that allows currency to flow only when some controlling input value is “true”. So, the transistor is overlaid on top of an arrow and has another input representing the control value:

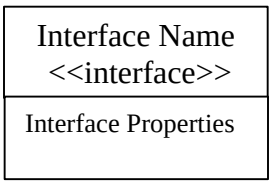


To illustrate game aspects other than gauges and their relationships, other diagramming techniques can be used. UML is particularly useful in this regard.

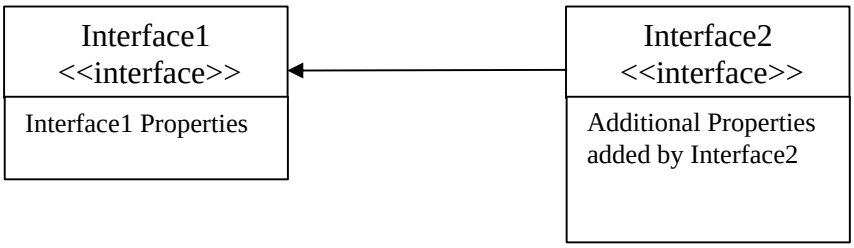
UML Component Diagram Illustrating the Interfaces supported by a component



UML Component Diagram illustrating the properties of an interface

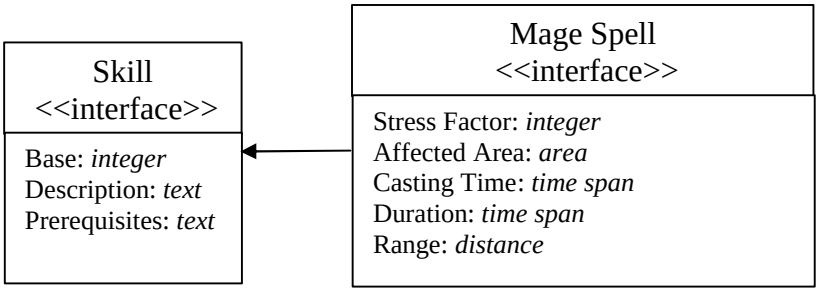


UML Component Diagram Illustrating "is a kind of" relationship



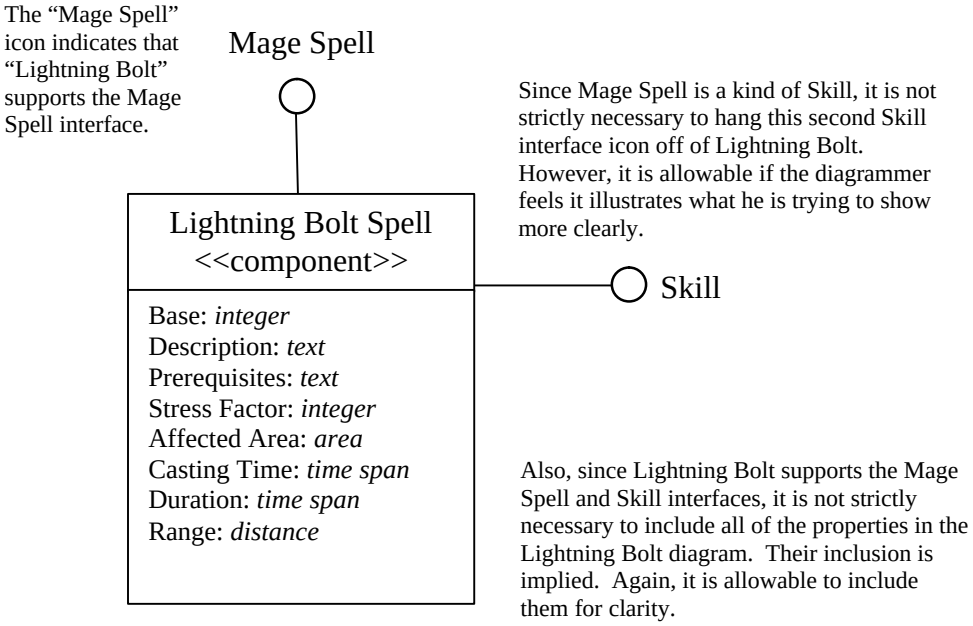
The arrow illustrates that Interface2 "is a kind of" Interface1. So, it inherits all of Interface1's properties.

UML Component Diagram Example of two interfaces



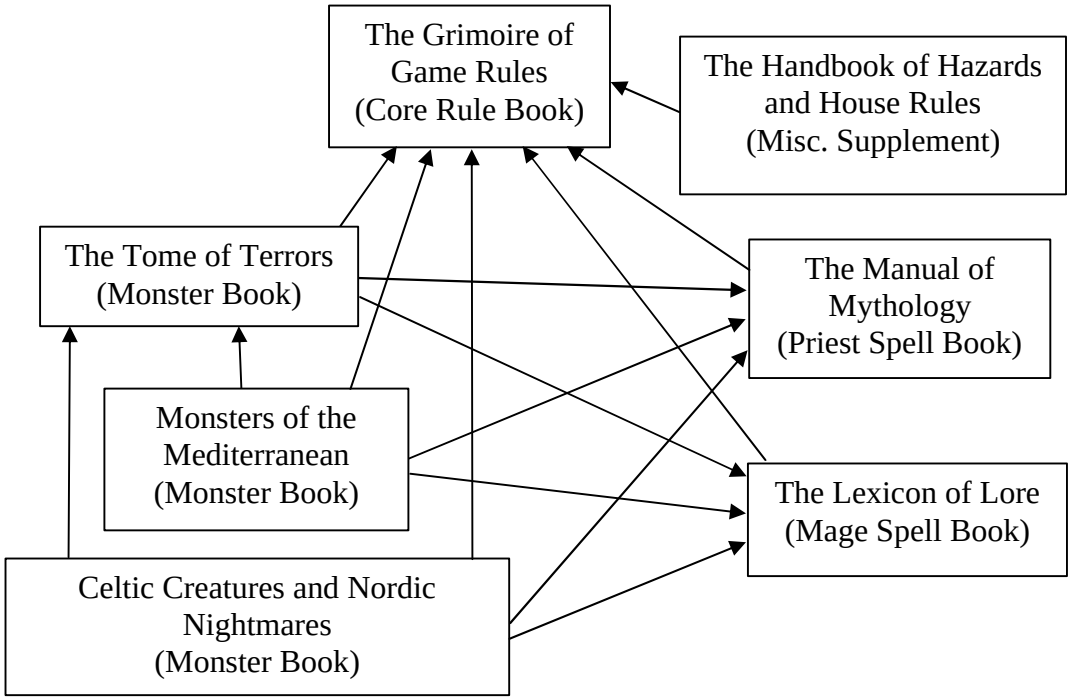
This illustrates that "Skill" and "Mage Spell" are both interfaces. The arrow indicates that "Mage Spell" is a kind of "Skill".

UML Example of a concrete component



UML Dependency Diagram

Example UML Dependency Diagram
(Illustrating the dependencies between the various books of Legendary Quest)



Strengths: There are no circular dependencies. It is therefore possible to play the game with *only* the Core Rule Book or a sub-set of the books. (The spell books, although not mandatory for play, are needed by the monster books.) The Core Rule Book has the responsibility of defining all of the interfaces used throughout the game. The supplements, then, merely implement concrete examples of these interfaces.

Weaknesses: Moving a small section out of The Tome of Terrors into The Grimoire of Game Rules would eliminate the dependencies on The Tome of Terrors entirely. In the next edition, I will do this. (This small section is, in essence, the definition of the "Monster" interface.)

Observation: All of the monster books depend on the two spells books as well as the Core Rule Book. This is because a decision was made to write up all specialized monster abilities as spells. This was a conscious decision I made to increase re-use of rules at the expense of adding dependencies between books. (This follows the "Modularity" design pattern.)