GameMatcher

Suggesting compatible video games based on your laptop's hardware specifications

Actionable Knowledge Representation

D2 - Ontology & Competency Questions

Group Members:

Andreea Scrob
Edoardo Tommasi

1 Refinements and Architectural Corrections

Following the feedback received on the initial domain analysis (D1), specifically regarding the lack of relations to model component connections and the inability to derive requirements fulfillment automatically, we have significantly refined the ontological model. These changes ensure compliance with ontology engineering best practices and transition the system from a purely descriptive model to an actionable knowledge base capable of automatic reasoning.

The complete implementation of the ontology is available at the following Google Colab.

1.1 Class Hierarchy Refinement: The GameAttribute Class

To improve the logical organization and maintainability of the ontology, we introduced an abstract superclass named `GameAttribute`, which serves as the parent concept for `GameGenre`, `PlayerMode`, and `PEGIRating`.

This structural refinement provides two key benefits:

- **Conceptual Grouping:** It formally categorizes these three distinct concepts as "qualitative features" of a video game, distinguishing them from physical hardware components or quantitative requirements.
- **Extensibility and Querying:** By grouping these classes under a common parent, the system gains the ability to perform generalized queries (e.g., *"Retrieve all attributes associated with Game X"*) without needing to query each subclass individually.

1.2 Structural Consolidations: Object Properties

To address the disconnect between entities, we formalized the topological structure of the graph by introducing key **Object Properties**:

- `hasComponent`: Defined as a **generic super-property**, it serves as the parent relation for all specific hardware connections (`hasCPU`, `hasGPU`, etc.). This hierarchical structure allows the system to query the `Computer` for any constituent part without needing to specify the component type, solving the issue of abstracting the machine's hardware composition.
- `hasMinRequirement`: It explicitly links a `VideoGame` to its `MinimumRequirement`, allowing the reasoner to navigate directly from a game instance to its technical constraints.

1.3 Transition to Quantitative Reasoning via Data Properties

A critical limitation in the initial design was the use of Object Properties for requirements (e.g., `requiresCPU`, `requiresGPU`). This approach forced a reasoning based on exact instance matching (e.g., a game requiring specifically an "Intel i5" would not automatically accept an "Intel i9").

To resolve this, we **replaced** these specific object properties with a set of **Data Properties** mapped to concrete XML Schema Datatypes (e.g., `xsd:float`, `xsd:int`). Instead of checking if a user owns a specific model of CPU, the system now compares measurable attributes.
Furthermore, we refined the modeling of storage: initially conceived as an abstract class (`AvailableStorage`), it has been correctly redefined as a physical component class (`Storage`), where the available capacity is now quantified directly via a specific Data Property.

Table 1 details the defined properties used to perform arithmetic comparisons (e.g., $\geq$, $\leq$) and string matching.

Table 1: Complete list of Data Properties for Quantitative Reasoning

Domain Class	**Data Property**	**Range**	**Description**
`CPU`, `GPU`	`hasBenchmarkScore`	`xsd:int`	User's hardware performance score.
`MinimumRequirement`	`requiresBenchmarkScore`	`xsd:int`	Threshold required by the game.
`RAM`, `GPU`	`hasMemorySizeGB`	`xsd:float`	Available memory (RAM or VRAM).
`MinimumRequirement`	`requiresMemoryGB`	`xsd:float`	Memory required.
`Storage`	`hasStorageSizeGB`	`xsd:float`	Free disk space on the specific storage component (HDD/SSD).
`MinimumRequirement`	`requiresStorageSpaceGB`	`xsd:float`	Game installation size.
`OperatingSystem`	`hasOSVersionValue`	`xsd:float`	Numeric version of the installed OS.
`MinimumRequirement`	`requiresMinOSVersionValue`	`xsd:float`	Minimum version needed.
`User`	`hasBudget`	`xsd:float`	Max price the user is willing to pay.
`VideoGame`	`hasPrice`	`xsd:float`	Cost of the game.
`User`	`hasAge`	`xsd:int`	User's chronological age.
`PEGIRating`	`hasPEGIAgeThreshold`	`xsd:int`	Minimum age for content.

Reasoning Example: Consider the following compatibility scenario involving a user and a game ("Cyberpunk"). The reasoning engine performs numerical comparisons rather than semantic matches:

- **Hardware Check:** The User's CPU has a `hasBenchmarkScore = 25000`. The Game's requirement specifies a `requiresBenchmarkScore = 15000`. $25000 \geq 15000 \implies$ `True`

- **Software Check:** The User runs Windows 11 (`hasOSVersionValue = 11.0`). The game requires a minimum of Windows 10 (`requiresMinOSVersionValue = 10.0`). $11.0 \geq 10.0 \implies$ `True`
- **Suitability Check:** The User is 25 years old (`hasAge = 25`). The game is rated PEGI 18 (`hasPEGIAgeThreshold = 18`). $25 \geq 18 \implies$ `True`

Since all logical conditions are met, the system infers the derived relation `isCompatibleWith`.

2 Competency Questions

The following list details the functional requirements of the GameMatcher ontology. Each entry defines a specific question that the system must be able to answer based on the modeled knowledge.

CQ1 **Hardware Compatibility Check**
Question: Which video games can run on the computer owned by User X?

CQ2 **Budget Feasibility Check**
Question: Can User X afford to buy VideoGame Y?

CQ3 **PEGI Age Compliance**
Question: Is User X old enough to play VideoGame Y based on the PEGI rating?

CQ4 **Genre Preference Matching**
Question: Which video games belong to the genres preferred by User X?

CQ5 **Player Mode Compatibility**
Question: Does VideoGame Y support the player mode preferred by User X?

CQ6 **RAM Requirement**
Question: Which video games require more than Z[1] GB of RAM?

Technical Categorization of Competency Questions

Table 2 categorizes each Competency Question according to its query type, predicate arity, relation type, and domain independence, as required by the ontology engineering guidelines.

[1] for example 8 or 16.

Table 2: Categorization of Competency Questions

ID	Topic	Type	Arity	Relation Type	Dom-Indep.
CQ1	Hardware Check	Selection	2	ObjectProp.	No
CQ2	Budget Check	Binary	2	DatatypeProp.	No
CQ3	Age Compliance	Binary	2	DatatypeProp.	No
CQ4	Genre Match	Selection	2	ObjectProp.	No
CQ5	Player Mode	Binary	2	ObjectProp.	No
CQ6	RAM Requirement	Selection	2	DatatypeProp.	No