Ontologie einer Rechnerarchitektur

Grundgedanke

Diese Ontologie soll eine **praktische, wiederverwendbare und erweiterbare Abbildung** einer **Rechnerarchitektur** sein.

Grundlegend unterteilt sich dabei ein Rechner (hier gleichzusetzen mit Computer) in viele einzelne Computer-Teile, die stets als „Teil“, veranschaulichend für eine Sache und gleichzeitig Untergruppe des komplexen Systems, benannt sind.

Grundlegender Aufbau

Da jede Untergruppe des Computers ein Teil des Computers ist, entspringen die sekundären Gruppen „**Hardware-Teil**“, „**Software-Teil**“ und „**Peripherie-Teil**“ der primären Übergruppe „Computer-Teil“.

Hardware-Teil

In der Hardware werden die einzelnen Komponenten eines Computers behandelt. Ein wichtiger Bestandteil ist die **CPU** mit dem **Register-Teil** und dem **ALU-Teil** (arithmetisch logische Einheit für Rechenoperationen). Als Teil eines (Daten-)Registers ist der Akkumulator für die Zwischenspeicherung von Ergebnissen der **ALU** zuständig, benötigt jedoch selbst auch Addierer- und Multiplizierer-Einheiten. Somit ergeben sich für dieses Teil folgende Eigenschaften:

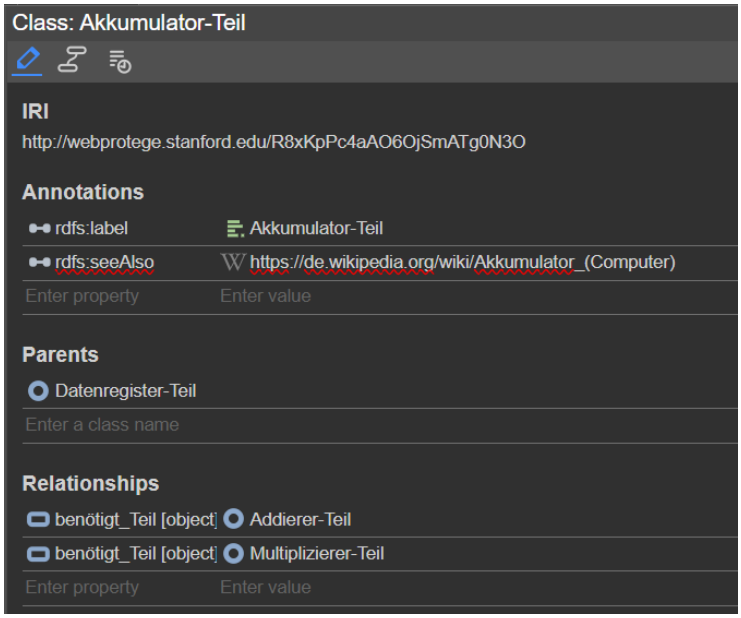

Abbildung 1: Eigenschaften des Akkumulator-Teils. "benötigt_Teil" ist ein object property und zeigt Objektabhängigkeiten.

Interessant dabei ist der benötigte Addierer-Teil; anschaulich ist der **Voll-Addierer** mit seinen vollen Abhängigkeiten und Überklassen in der Abbildung 2 zu sehen. Der Voll-Addierer benötigt für seine volle Funktionalität einen Halbaddierer (inkl. AND- und XOR-Gatter), ein NOT- (Negation) und ein XNOR-Gatter (Äquivalenz) mit insgesamt 3 Eingängen (x_0, y_0, c_0) und 2 Ausgängen (z_0, c_1). Die resultierende Darstellung ist beachtlich komplex.

Das Kühlsystem-Teil in der jeweiligen Ausführung, wie Gebläsekühler oder Passivkühler, wird für das Innenleben des Rechners selbst und für den Grafikprozessor in der Grafikkarte gebraucht. Weitere Klassen sind der Ontologie zu entnehmen.

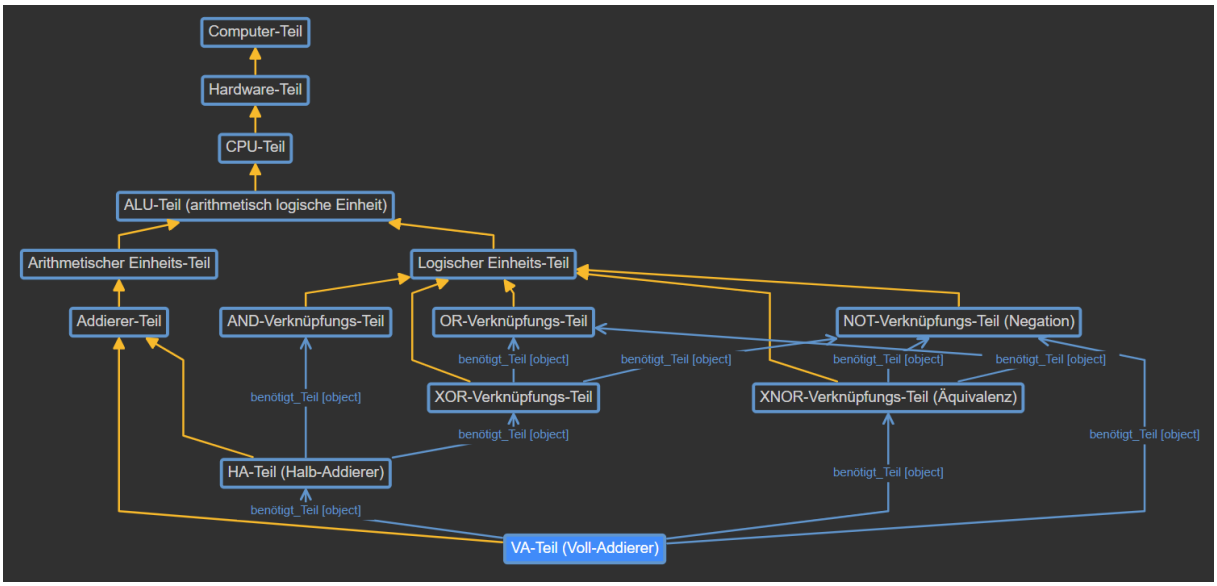

Abbildung 2: Die gesamte Darstellung eines VA mit Abhängigkeiten und Parentklassen.

Software-Teil

Die Software gliedert sich in **Anwendungssoftware und Systemsoftware** auf. Hierauf lassen sich viele der Data-Properties anwenden.

Peripherie-Teil

Die Peripherie ist nach **Eingabe-, Ausgabe- und Hybrid-Teil** gebündelt, die über die Richtung des Datenflusses Auskunft geben. Gerade bei komplizierteren Geräten, wie den 3D-Druckern (zugehörig zur permanent-sichtbaren Ausgabe) wurde hier nicht auf Beschreibungen (direkt in der Klasse) und Quellen (in der Überklasse) verzichtet:

> *Digital Light Processing (DLP): Funktionsweise: Aushärtung von Harz mit digitaler Lichtprojektion. Anwendungen: Ähnlich wie SLA, aber schneller. Vorteile: Hohe Auflösung, schnelle Druckzeiten. Nachteile: Begrenzte Materialauswahl, teure Drucker.*

Individuals

Als Individuen wurde folgendes Szenario nachgestellt: Für einen **Endverbraucher-PC** wurden alle benötigten **Hardware-Komponenten** (von CPU bis SSD) ausgewählt und mit den wichtigsten Details (Hersteller, Kosten, Produkt Id, …) und produktabhängigen Eigenschaften (Anzahl der Kerne, Taktfrequenz, …) versehen. Als Peripherie wurde neben dem **Standard** (Maus, Tastatur, Bildschirm) noch ein Fused Deposition Modeling **3D-Drucker** als permanent sichtbare Ausgabe hinzugefügt. Auf dem PC soll dann **Microsoft** mit MS Word als Produktivitätssoftware und Teams für die Live-Kommunikation laufen. Die Individuen sind in Abbildung 3 zu sehen, Abbildung 4 zeigt den 3D-Drucker als ausgewähltes Individuum.

Abbildung 3: Die ausgearbeiteten Individuen

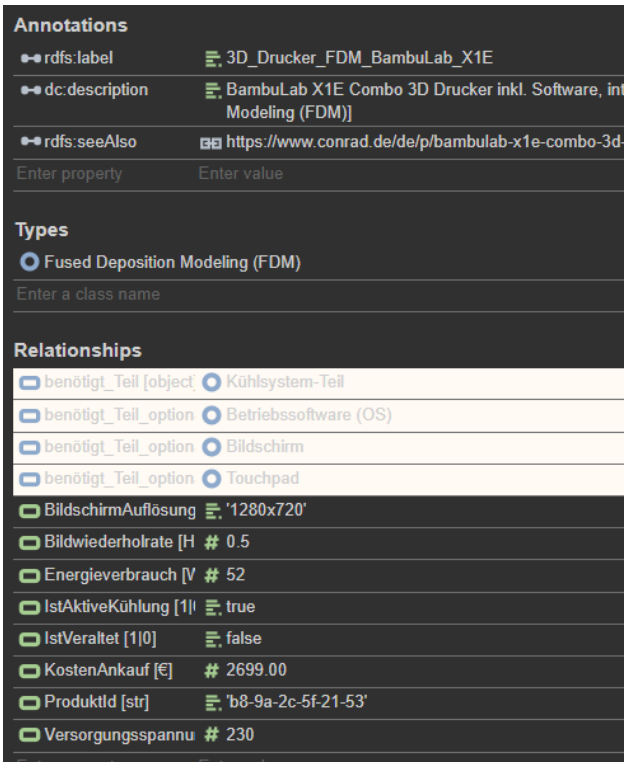

Abbildung 4: Der Fused Deposition Modeling 3D-Drucker. Die von der UI weiß hinterlegten Zeilen zeigen die Object Properties „benötigt_Teil“ und „benötigt_Teil_optional“

Properties

Neben dem **Object-Property** „**benötigt-Teil**“ für **Subsumptionen** beinhaltet die Ontologie auch weitere Data-Properties, wie folgender Abbildung 5 dargestellt.

Jedes **Data-Property** beinhaltet im Namen die Eigenschaft in CamelCase und die Einheit, bzw. die eintragbaren Werte in Klammern. Zudem ist jede Property einer oder mehrerer Domains zugeordnet und trägt einen festen Typen als Range. Hier ist als Beispiel das Data-Property „Energieverbrauch [W]“ dargestellt:

> *rdfs:label: Energieverbrauch [W]*
>
> *dc:description: ALU-Energieverbauch: Der Energieverbrauch einer arithmetisch-logischen Einheit (ALU) kann stark variieren […]. CPU-Energieverbrauch: - Desktop-CPUs: Typischerweise zwischen 65 und 125 Watt (TDP - Thermal Design Power). […]*
>
> *Domain: Peripherie-Teil, ALU-Teil (arithmetisch logische Einheit), Grafikkarten-Teil, CPU-Teil*
>
> *Range: xsd:double*

Somit kann jedem Data-Property ein nachvollziehbarer Wert zugewiesen werden.

Abbildung 3: Einige Data-Properties zur Ontologie

Abschließende Worte

Das Projekt ist dafür aufgebaut, weiter **von außen erweiterbar** zu sein.

Die Klassen Ontologie_Besitzer, Ontologie_Ortsabhängigkeit, Ontologie_Aktivität und Ontologie_Zeitanhängigkeit sind vom **DFKI Ubisworld** inspiriert, sodass hier die Ontologie mit anderen Ontologien erweitert werden könnte. Leider konnte dafür die **UbisWorld 3.0**, wie in einem Paper[1] beschrieben, nicht über die dargestellte Website[2] aufgerufen werden.

Das Projekt der Rechnerarchitektur kann über den folgenden Link aufgerufen werden: https://webprotege.stanford.edu/#projects/d74a8091-3c97-4ef0-92f5-a775aa0c3ab2

Es ist zudem auf GitHub zu finden: https://github.com/s-voelkl/Computer-Architecture-OWL-Ontology

[1] Heckmann, Dominikus, et al.: „UbisWorld 3.0: a semantic Tool Set for Ubiquitous User Modeling“, DFKI GmbH. https://www.dfki.de/fileadmin/user_upload/import/4322_UbisWorldSystem.submitted.pdf

[2] Link zur Website: http://www.ubisworld.org/ liefert einen Proxy Error mit der Begründung „DNS lookup failure for: ubisworld.ai.cs.uni-sb.de“.