ACH2018

Gustavo Tsuyoshi Ariga

November 19, 2023

Contents

Chapter 1

Segundo Semestre

Aula 1: Planejamento 17 Julho 2023

1.1 Semana 01

A integração do WoTPy com grafos de conhecimento pode ser realizada utilizando a semântica do WoT para descrever dispositivos IoT e suas interações, e em seguida representar essas informações em um grafo de conhecimento.

1.1.1 Descrição dos dispositivos e interações utilizando o WoT

O WoT oferece uma abordagem padronizada e unificada para descrever e interagir com dispositivos e serviços da Internet das Coisas (IoT). Essa abordagem utiliza o conceito de "Coisas", que são representadas por Descrições de Coisas (TDs).

Cada TD é um documento baseado em JSON-LD que fornece metadados sobre uma "Coisa", incluindo seu nome, descrição, localização, metadados de segurança e as interações disponíveis, que podem ser propriedades, ações e eventos.

Propriedades representam o estado de uma "Coisa", como a temperatura atual de um sensor de temperatura ou o status de uma lâmpada (ligada/desligada). Algumas propriedades podem ser lidas e outras podem ser escritas, permitindo a alteração do estado da "Coisa".

Ações representam funções que podem ser executadas em uma "Coisa". Por exemplo, é possível ter uma ação para redefinir um dispositivo ou iniciar uma máquina de lavar. As ações são invocadas por meio de solicitações enviadas à "Coisa", que então executa a ação e retorna uma resposta.

Eventos representam notificações enviadas por uma "Coisa". Por exemplo, um

sensor de porta pode enviar um evento quando a porta é aberta, ou uma lâmpada pode enviar um evento quando seu status muda. Os eventos são enviados da "Coisa" para os consumidores que se inscreveram para recebê-los.

Com o WoTPy, é possível criar um servidor WoT que hospeda várias "Coisas" e seus TDs. Esse servidor expõe uma interface de rede (geralmente HTTP, mas também pode ser CoAP, MQTT, etc.) que permite aos consumidores descobrir as "Coisas", obter seus TDs e interagir com elas.

Para criar uma "Coisa" com o WoTPy, é necessário criar uma instância da classe "Thing" e adicionar propriedades, ações e eventos a ela. Em seguida, a classe "Servient" pode ser utilizada para criar um servidor WoT que hospeda a "Coisa". A classe "Servient" também oferece métodos para iniciar e parar o servidor.

Além disso, o WoTPy também permite a criação de clientes WoT, que podem consumir TDs e interagir com as "Coisas" por meio deles. O cliente WoT pode ser utilizado para descobrir "Coisas" em um servidor WoT, obter os TDs das "Coisas" e ler/escrever propriedades, invocar ações e se inscrever para receber eventos.

Para criar um cliente WoT com o WoTPy, é possível utilizar a classe "ConsumedThing", que fornece métodos para interagir com uma "Coisa" com base em seu TD. Essa classe é utilizada em conjunto com a classe "Servient", que oferece métodos para criar e gerenciar os consumidores WoT.

1.1.2 Criação do Grafo de Conhecimento

Um grafo de conhecimento é um modelo de dados que representa entidades (nós) e as relações entre elas (arestas). No contexto do Web of Things (WoT), essas entidades podem ser as "Coisas" (dispositivos IoT) e suas interações, como propriedades, ações e eventos.

Para criar um grafo de conhecimento a partir das descrições do WoT, é necessário identificar as entidades e as relações a partir dos Thing Descriptions (TDs). Cada TD contém informações que podem ser mapeadas para nós e arestas do grafo.

Nós: Cada "Coisa" descrita em um TD pode ser considerada um nó no grafo. Além disso, cada propriedade, ação e evento de uma "Coisa" também pode ser considerado um nó separado. Por exemplo, se houver uma "Coisa" que seja um sensor de temperatura, pode-se ter um nó para o sensor, um nó para a propriedade de leitura de temperatura e um nó para o evento de notificação de temperatura excedida.

Arestas: As relações entre as "Coisas" e suas interações podem ser representadas como arestas no grafo. As arestas devem indicar a direção da relação. Por exemplo, pode-se ter uma aresta do nó do sensor para o nó de leitura de temperatura, indicando que o sensor "tem uma propriedade" de leitura de tem-

peratura. Da mesma forma, pode-se ter uma aresta do nó do sensor para o nó de notificação de temperatura excedida, indicando que o sensor "pode gerar um evento" de notificação de temperatura excedida.

Uma vez identificados os nós e as arestas, é possível utilizar uma biblioteca de manipulação de grafos para criar o grafo de conhecimento. Existem diversas bibliotecas de Python disponíveis para essa finalidade, como:

- NetworkX: É uma biblioteca de Python para criação, manipulação e estudo de redes complexas. O NetworkX permite a criação de grafos direcionados e não direcionados, além de oferecer algoritmos para cálculo de caminhos mais curtos, detecção de comunidades, centralidade, entre outros.
- graph-tool: É uma biblioteca de Python para manipulação e análise de grafos. O graph-tool oferece funcionalidades semelhantes ao NetworkX, mas é otimizado para eficiência com grafos grandes.
- PyGraphviz: É uma interface Python para a biblioteca Graphviz, que é utilizada para criar visualizações de grafos. O PyGraphviz permite a criação, manipulação e desenho de grafos, mas não oferece muitos algoritmos para análise de grafos.

Ao escolher uma biblioteca, é importante considerar as necessidades específicas, como o tamanho do grafo, a complexidade das análises que serão realizadas e se é necessário ou não recursos de visualização.

1.1.3 Adição de Semântica ao Grafo de Conhecimento

A adição de semântica ao grafo de conhecimento envolve a atribuição de significados aos nós e arestas que sejam compreendidos não apenas por humanos, mas também por máquinas. Essa semântica permite que as máquinas entendam e interpretem os dados do grafo, facilitando a integração, busca, análise e inferência de dados.

Para adicionar semântica ao grafo, é necessário atribuir a cada nó e aresta um tipo de uma ontologia. Uma ontologia é uma descrição formal de um domínio de conhecimento que inclui a definição dos conceitos desse domínio (chamados de classes), as propriedades desses conceitos (chamadas de propriedades de objetos ou de dados) e as restrições sobre essas propriedades.

Por exemplo, é possível utilizar a ontologia SSN (Semantic Sensor Network) para adicionar semântica aos nós e arestas que representam sensores e suas interações. A SSN inclui classes como "Sensor", "Observation", "Property" e "Event", além de propriedades como "observes", "hasProperty" e "detects".

Para adicionar semântica ao grafo de conhecimento com a SSN, é possível seguir as seguintes etapas:

1. Atribuir a classe "Sensor" ao nó que representa o sensor de temperatura.

2. Atribuir a classe "Property" ao nó que representa a leitura de temperatura.
3. Atribuir a propriedade "hasProperty" à aresta que conecta o sensor à leitura de temperatura.
4. Atribuir a classe "Event" ao nó que representa a notificação de temperatura excedida.
5. Atribuir a propriedade "detects" à aresta que conecta o sensor à notificação de temperatura excedida.

Existem diversas ferramentas e bibliotecas disponíveis para adicionar semântica ao grafo de conhecimento, como RDFlib (uma biblioteca de Python para trabalhar com RDF, uma linguagem padrão para representar grafos de conhecimento semântico), Protégé (uma ferramenta de edição e gerenciamento de ontologias) e Apache Jena (uma estrutura de programação para construir aplicações de Web Semântica).

Após adicionar semântica ao grafo de conhecimento, é possível utilizar uma linguagem de consulta como SPARQL para realizar consultas complexas e inferências nos dados. Além disso, a semântica permite a integração do grafo com outros grafos de conhecimento semântico, melhorando a interoperabilidade e possibilitando a reutilização de dados.

1.1.4 Utilização do Grafo de Conhecimento

Consultas SPARQL: A linguagem de consulta SPARQL permite realizar consultas complexas no grafo de conhecimento. É possível utilizar SPARQL para encontrar todas as "Coisas" que possuem uma determinada propriedade, ou para encontrar todas as ações que podem ser invocadas em um determinado conjunto de "Coisas". Além disso, é possível realizar consultas que envolvem múltiplas entidades e relações, por exemplo, encontrar todas as "Coisas" que podem observar uma propriedade específica e gerar um evento específico.

Inferência de novas relações: Ao analisar o grafo de conhecimento, é possível inferir novas relações que não foram explicitamente declaradas. Por exemplo, se duas "Coisas" possuem várias propriedades e ações em comum, é possível inferir que elas são de alguma forma semelhantes ou relacionadas. Da mesma forma, se uma "Coisa" possui uma ação que afeta a propriedade de outra "Coisa", é possível inferir que existe uma relação causal entre as duas "Coisas".

Melhor interoperabilidade: Uma vez que o grafo de conhecimento codifica a semântica das "Coisas" e suas interações, ele permite que os clientes que compreendem essa semântica interajam de maneira mais eficiente com as "Coisas". Por exemplo, um cliente pode utilizar o grafo para descobrir como interagir com uma nova "Coisa" que nunca encontrou antes, simplesmente analisando suas propriedades, ações e eventos e comparando-os com os de outras "Coisas" conhecidas. Além disso, a semântica permite que o cliente compreenda o significado das interações, o que pode ajudar a evitar erros e melhorar a qualidade

da interação.

Integração e reutilização de dados: O grafo de conhecimento semântico facilita a integração de dados provenientes de diferentes fontes e a reutilização de dados em diferentes contextos. Como as entidades e relações no grafo são anotadas com tipos de uma ontologia, é mais fácil mapear e combinar dados de diferentes grafos que utilizam a mesma ontologia ou ontologias compatíveis. Além disso, os dados anotados semanticamente são mais fáceis de compreender e reutilizar, pois seu significado é explicitamente declarado e codificado de forma padronizada.

1.2 Semana 02

Aula 2: Definição dos Conceitos

24 Julho 2023

A integração proposta envolve a utilização da semântica do WoT (Web of Things) para descrever os dispositivos IoT e suas capacidades, e a representação dessas informações em um grafo de conhecimento, enriquecido com ontologias como a SSN (Semantic Sensor Network) e a SOSA (Sensor, Observation, Sample, and Actuator).

1.2.1 Plano de Atividades

Capacitação do Aluno para o Ferramental de Grafos de Conhecimento

Antes de iniciar o processo de integração, é fundamental que o aluno se familiarize com as ferramentas e conceitos relacionados aos grafos de conhecimento. Para isso, são recomendadas as bibliotecas RDFLib e OWLReady, que permitem a manipulação de RDF (Resource Description Framework) e ontologias em Python.

Escolha da Plataforma de Armazenamento de Grafos de Conhecimento

Uma vez capacitado, o próximo passo é selecionar uma plataforma ou biblioteca para o armazenamento e manipulação dos grafos de conhecimento. As bibliotecas RDFLib e OWLReady, previamente mencionadas, podem ser utilizadas para essa finalidade.

Armazenamento das Capacidades do Sensor Codificadas em SSN e Observações em SOSA

As informações sobre os dispositivos IoT e suas capacidades são descritas utilizando a semântica do WoT e a ontologia SSN. Além disso, as observações realizadas pelos sensores são codificadas utilizando a ontologia SOSA. Essa representação semântica permite que os dados sejam interpretados tanto por humanos quanto por máquinas, facilitando a compreensão e a interoperabilidade.

Publicização do Grafo de Conhecimento através de um Endpoint SPARQL

Para que os dados do grafo de conhecimento sejam acessíveis e consultáveis, é disponibilizado um endpoint SPARQL. Esse endpoint permite que os clientes realizem consultas ao grafo de conhecimento, obtendo informações específicas sobre os dispositivos IoT e suas interações.

Definição e Implementação de Casos de Teste da Integração

Para garantir a qualidade da integração, é fundamental definir e implementar casos de teste que validem o correto funcionamento do sistema integrado. Os casos de teste devem abranger diferentes cenários de interação com os dispositivos IoT e verificar se as respostas e resultados estão de acordo com as expectativas.

1.2.2 Detalhamento

Familiarização com Grafos de Conhecimento: Antes de iniciar a integração, é importante que os desenvolvedores envolvidos se familiarizem com os conceitos e ferramentas relacionados aos grafos de conhecimento. Isso inclui o entendimento de ontologias, RDF (Resource Description Framework) e outras tecnologias semânticas.

Escolha de uma Plataforma de Armazenamento de Grafos: O próximo passo é selecionar uma plataforma ou biblioteca que suporte o armazenamento e manipulação de grafos de conhecimento. Algumas opções populares incluem RDFLib, OWLReady e GrahpDB. A escolha deve levar em consideração a escalabilidade e as necessidades específicas do projeto.

Descrição dos Dispositivos e Interações usando WoTPy: Utilize o WoTPy para descrever os dispositivos IoT e suas interações, utilizando a semântica do WoT. Cada dispositivo deve ser representado por uma "Coisa" com suas propriedades, ações e eventos. É importante que essas descrições sejam detalhadas e padronizadas para garantir a correta representação no grafo de conhecimento.

Mapeamento para Ontologias: Mapeie as descrições dos dispositivos e interações feitas em WoTPy para as ontologias relevantes, como a SSN (Semantic Sensor Network) e a SOSA (Sensor, Observation, Sample, and Actuator). Essas ontologias fornecem uma estrutura semântica comum para a representação dos dispositivos IoT no grafo de conhecimento.

Armazenamento no Grafo de Conhecimento: Utilize a plataforma escolhida para armazenar as informações dos dispositivos IoT e suas interações no grafo de conhecimento, seguindo as estruturas e relações definidas pelas ontologias mapeadas.

Criação de um Endpoint SPARQL: Crie um endpoint SPARQL que permita a consulta e acesso aos dados do grafo de conhecimento. O endpoint SPARQL é essencial para que os clientes possam realizar consultas específicas sobre os dispositivos e interações.

Implementação de Casos de Teste: Defina casos de teste para validar a integração do WoTPy com o grafo de conhecimento. Os casos de teste devem

abranger cenários típicos de interação e garantir que as respostas e resultados estejam corretos e de acordo com as expectativas.

Realizar Testes e Ajustes: Execute os casos de teste e verifique se a integração está funcionando conforme o esperado. Realize ajustes e correções, se necessário, para garantir a qualidade e precisão dos dados no grafo de conhecimento.

Uso e Exploração do Grafo de Conhecimento: Após a integração bem-sucedida, o grafo de conhecimento estará disponível para uso e exploração. Os clientes podem realizar consultas SPARQL para obter informações específicas sobre os dispositivos IoT e suas interações, aproveitando os benefícios da representação semântica e a interoperabilidade proporcionada pela integração do WoTPy com grafos de conhecimento.

1.2.3 Conceitos Fundamentais

Modelo RDF (Resource Description Framework)

O RDF é um modelo padrão da Web Semântica que permite representar informações na forma de triplas. Ele fornece uma maneira de descrever recursos (entidades) e suas propriedades em formato de sentenças simples e declarativas. Cada sentença é composta por três partes principais: o sujeito, o predicado e o objeto.

- Sujeito: Representa o recurso ou entidade a ser descrita. É representado por um URI (Identificador Uniforme de Recurso) que identifica unicamente o recurso.
- Predicado: Representa a relação ou propriedade que liga o sujeito ao objeto. Também é representado por um URI e denota um tipo de relação entre os dois.
- Objeto: Representa o valor da propriedade ou outra entidade relacionada ao sujeito. Pode ser um URI, um literal (valor de dados) ou outro recurso.

Por exemplo, a sentença "João é um Engenheiro" pode ser representada em RDF como:

- Sujeito: URI do recurso que representa "João".
- Predicado: URI do predicado que representa "é um".
- Objeto: URI do recurso que representa "Engenheiro".

Estrutura de Triplas

As triplas são a base do modelo RDF e formam a estrutura fundamental dos grafos de conhecimento. Elas representam as afirmações sobre os recursos e suas propriedades. Várias triplas juntas criam o grafo de conhecimento, em que os nós são os recursos e as arestas são os predicados que conectam os recursos. O formato básico das triplas é: (Sujeito, Predicado, Objeto)

Por exemplo, representando informações sobre uma pessoa chamada "Maria":

- Sujeito: URI representando "Maria".
- Predicado: URI representando "nome".
- Objeto: Literal representando o valor "Maria".

Essa tripla "Maria" - "nome" - "Maria" afirma que o nome da pessoa é "Maria".

O RDF é altamente flexível, permitindo a criação de grafos de conhecimento com informações estruturadas e conectadas de maneira eficiente. Através da combinação de várias triplas, é possível representar dados complexos e relacionamentos entre entidades.

Estudar e compreender esses conceitos é essencial para se aprofundar na construção e manipulação de grafos de conhecimento, bem como para tirar o máximo proveito dessa poderosa tecnologia na Web Semântica e em outros domínios de aplicação.

Ontologias

Ontologias são esquemas formais que definem a estrutura, os conceitos, as propriedades e as relações de um domínio específico. Elas descrevem um vocabulário comum e compartilhado que permite uma compreensão comum entre humanos e máquinas. As ontologias são fundamentais para adicionar significado e semântica aos dados em um grafo de conhecimento. Por exemplo, uma ontologia para o domínio de saúde pode definir conceitos como "Paciente", "Médico", "Doença" e propriedades como "temIdade", "temDiagnóstico". Essas definições claras permitem que os dados relacionados a esses conceitos sejam interpretados corretamente e que consultas e inferências sejam realizadas de maneira precisa.

RDFS (RDF Schema) e OWL (Web Ontology Language): RDFS e OWL são linguagens de modelagem usadas para criar ontologias no RDF. RDFS é uma linguagem mais simples, que permite a definição de hierarquias de classes, propriedades e inferências básicas. Já o OWL é mais poderoso e permite a criação de ontologias mais expressivas, incluindo restrições complexas, definição de equivalência e inferências avançadas.

Inferência

A inferência é a capacidade de um sistema de realizar deduções lógicas e conclusões a partir dos dados disponíveis no grafo de conhecimento. Com base nas regras definidas nas ontologias e no raciocínio lógico, é possível inferir novas informações que não estão explicitamente declaradas no grafo.

Por exemplo, se o grafo contém a informação de que "Maria é filha de João" e "João é filho de Ana", o sistema pode inferir que "Maria é neta de Ana" utilizando a ontologia que define a relação de parentesco.

Endpoint SPARQL

O SPARQL é a linguagem de consulta utilizada para recuperar informações de um grafo de conhecimento. Um endpoint SPARQL é um serviço ou ponto de acesso que permite enviar consultas SPARQL e receber os resultados correspondentes. Esse recurso é fundamental para permitir que aplicativos e sistemas externos interajam com o grafo de conhecimento e realizem consultas complexas para obter informações específicas.

Linked Data

O conceito de Linked Data (Dados Ligados) refere-se à prática de conectar e interligar grafos de conhecimento através de URIs para criar uma rede global de dados estruturados na Web. Isso permite que diferentes grafos de conhecimento sejam combinados e enriquecidos com informações de outras fontes, aumentando a eficácia, a reutilização e a interoperabilidade dos dados.

Ingestão e Atualização de Dados

A ingestão de dados refere-se ao processo de adicionar informações ao grafo de conhecimento, enquanto a atualização é o processo de modificar ou remover informações existentes. Esses processos são essenciais para manter o grafo de conhecimento atualizado e relevante ao longo do tempo, à medida que novos dados são disponibilizados.

1.2.4 Próximos Passos

já se tem uma configuração funcional em que o sensor ESP32 lê os valores do sensor UV e os envia para o servidor WoT. O servidor WoT, por sua vez, expõe esses dados como propriedades de uma "Coisa" na terminologia WoT.

Para criar um grafo de conhecimento com essas informações, é necessário seguir os seguintes passos:

Extração de dados do servidor WoT: Os dados do servidor WoT já estão configurados e funcionais. O próximo passo é extrair esses dados. Os dados UV enviados pelo sensor ESP32 para o servidor WoT são acessíveis através das propriedades da "Coisa" exposta pelo servidor WoT. Esses dados serão usados para construir o grafo de conhecimento.

Criação de um modelo RDF/OWL: Deve-se criar um modelo RDF ou OWL para representar as "Coisas" WoT e suas propriedades. Por exemplo, pode-se ter uma classe "Dispositivo" com subclasses específicas como "SensorUV" e propriedades como "temLeitura". O RDFlib ou OWLReady podem ser usados para criar esse modelo, de acordo com a complexidade do mesmo.

Povoamento do grafo de conhecimento: Os dados extraídos do servidor WoT serão utilizados para criar instâncias das classes e definir suas propriedades. Por exemplo, será criada uma instância da classe "SensorUV" para cada sensor

UV no sistema, e a propriedade "temLeitura" será definida para o valor de leitura mais recente.

Armazenamento e consulta do grafo de conhecimento: O grafo de conhecimento será armazenado em um banco de dados RDF ou Triplestore, como o RDFLib's IOMemory ou um banco de dados SPARQL completo, como o Virtuoso. Isso permitirá realizar consultas SPARQL complexas, fazer inferências e outras operações.

Ao seguir essas etapas, as informações do servidor WoT poderão ser representadas em um grafo de conhecimento, o qual poderá ser utilizado para realizar consultas e inferências, além de possibilitar a expansão do sistema IoT incorporando mais dispositivos e tipos de dados.

1.2.5 Criação do Grafo de Conhecimento RDF

O RDF (Resource Description Framework) é uma estrutura padrão para troca de dados na Web. Ele tem recursos que facilitam a mesclagem de dados, mesmo que os esquemas subjacentes sejam diferentes, e suporta especificamente a evolução dos esquemas ao longo do tempo sem a necessidade de alterar os dados.

No código, o grafo de conhecimento é construído usando a biblioteca rdflib, que é uma biblioteca Python para trabalhar com RDF. O grafo é uma maneira de modelar o conhecimento como uma rede de entidades e relacionamentos.

Aqui estão as partes relevantes do código:

Primeiro, um namespace é criado para as entidades do dispositivo:

```
n = Namespace("http://wotpyrdfsetup.org/device/")
```

Isso é usado para criar URLs únicas para as entidades no RDF. As URLs são importantes porque permitem que as entidades sejam referenciadas univocamente em todo o grafo.

Em seguida, as classes e propriedades são definidas no grafo:

```
Device = n["Device"]
UVSensor = n["UVSensor"]
hasReading = n["hasReading"]
sensor1 = n["sensor1"]
```

Aqui, Device e UVSensor são classes, hasReading é uma propriedade e sensor1 é uma instância de UVSensor.

Depois disso, as relações entre as entidades são adicionadas ao grafo:

```
g.add((UVSensor, RDFS.subClassOf, Device))
g.add((sensor1, RDF.type, UVSensor))
g.add((sensor1, hasReading, Literal(uv_data)))
```

A primeira linha está dizendo que UVSensor é uma subclasse de Device. A segunda linha está dizendo que sensor1 é do tipo UVSensor. A última linha está adicionando a leitura do sensor UV ao sensor1 usando a propriedade hasReading.

Finalmente, o grafo é serializado em formato "turtle" para depuração:

```
print(g.serialize(format="turtle"))
```

O resultado é uma representação gráfica dos dados do sensor em um formato que pode ser facilmente compartilhado e entendido por outras aplicações compatíveis com RDF.

1.3 Semana 03

Aula 3: SSN

21 Agosto 2023

Especificação do W3C para o "Semantic Sensor Network (SSN) Ontology", que é uma ontologia para descrever sensores e suas observações. A SSN é usada para representar diferentes aspectos de sistemas de sensoriamento, incluindo o hardware do sensor (como o próprio sensor e o sistema no qual ele está embutido), os processos de sensoriamento e os dados produzidos.

Os exemplos fornecidos no https://www.w3.org/TR/vocab-ssn/#examples mostram como usar a ontologia SSN para representar várias entidades e relacionamentos em um sistema de sensoriamento. Por exemplo, como representar um sensor, suas capacidades, a propriedade que ele observa, as condições sob as quais opera, etc.

1.3.1 endpoint SPARQL

Implementar um endpoint SPARQL no Tornado implica em criar um ponto de acesso em seu servidor Tornado que aceite consultas SPARQL e retorne resultados. Aqui está um exemplo básico de como você pode fazer isso, usando o Tornado junto com a rdflib para armazenar e consultar um grafo RDF.

Inicie o servidor e execute uma consulta SPARQL usando o parâmetro query na URL. Por exemplo: http://localhost:8888/sparql?query=SELECT%20%3Fsubject%20%3Fobject%20WHERE%20%7B%3Fsubject%20%3Fpredicate%20%3Fobject%20.%7D

1.3.2 SSN

Para implementar um endpoint SPARQL no Tornado que pode lidar com dados modelados usando a SSN, seria necessário uma abordagem semelhante à fornecida na resposta anterior, mas seu grafo RDF estaria preenchido com dados conforme a ontologia SSN.

Por exemplo, você pode ter um grafo RDF que contém informações sobre um termômetro, que é um sensor que mede a temperatura. Esse termômetro poderia

ser modelado usando a ontologia SSN para indicar coisas como a propriedade que ele mede (temperatura), seu alcance de operação (por exemplo, -50°C a 150°C), a unidade de medida que usa (Celsius), etc.

Então, quando alguém envia uma consulta SPARQL para o seu endpoint, eles podem perguntar coisas como "Listar todos os sensores que medem temperatura" ou "Quais são os alcances operacionais de todos os termômetros no sistema?", e o seu servidor Tornado retornaria os resultados apropriados, com base nos dados em seu grafo RDF.

1.3.3 Código

Ontologia SSN:

Adicionei os namespaces para as ontologias ssn e sosa, que são partes essenciais da "Semantic Sensor Network Ontology". ssn é a ontologia original, enquanto sosa é uma ontologia simplificada e alinhada com ssn.

Definições de Tipo e Classe:

Defini UVSensor como uma instância de ssn.System. Na ontologia SSN, um sistema pode ser qualquer coisa, desde um simples sensor até uma plataforma inteira de sensores.

Criei uma classe UVObservation que é uma subclasse de Observation. Isso permite modelar cada leitura UV como uma observação específica.

Defini UVValue como um datatype. Isso permite que valores UV sejam armazenados e semânticamente associados como resultados de observações.

Atualização do Grafo:

Na função write_uv, ajustei o método de atualização do grafo para usar conceitos da ontologia SSN.

Em vez de simplesmente armazenar uma leitura, agora modelamos uma "observação" realizada pelo sensor. Isso implica que o UVSensor "observa" uma UVObservation, que "tem um resultado" (HasResult) que é o valor UV.

A URI da observação é dinamicamente criada com base no valor UV, garantindo que cada observação seja única.

Removi as entradas antigas da observação antes de adicionar novas para evitar um acúmulo de observações antigas no grafo.

Serialização:

A serialização final do grafo em formato turtle permite que você veja como as informações são representadas usando a ontologia SSN. Essas mudanças garantem que os dados do sensor UV sejam representados de uma forma que esteja em conformidade com a ontologia SSN, permitindo uma integração mais rica e significativa com outros sistemas que compreendem e utilizam essa ontologia.

1.3.4 Reunião

Implementando uma Observação no Formato RDF

Uma observação em RDF (Resource Description Framework) refere-se a uma maneira estruturada de representar informações. Vamos destrinchar uma observação e entender o que cada parte dela significa:

```
<Observation/346345> rdf:type sosa:Observation ;
```

Aqui, estamos criando uma nova observação com o identificador <Observation/346345>. A relação rdf:type especifica que esta é uma instância da classe sosa:Observation.

```
sosa:observedProperty <sensor/35-207306-844818-0/BMP282/atmosphericPressure> ;
```

Esta tripla indica qual propriedade foi observada. Neste caso, a pressão atmosférica é a propriedade observada pelo sensor BMP282.

```
sosa:hasFeatureOfInterest  <earthAtmosphere> ;
```

Aqui, estamos especificando qual é o objeto ou entidade de interesse da observação, neste caso, a atmosfera terrestre.

```
sosa:madeBySensor <sensor/35-207306-844818-0/BMP282> ;
```

Indica qual sensor fez a observação. Neste caso, é o sensor BMP282.

```
sosa:hasResult [
  rdf:type qudt-1-1:QuantityValue ;
  qudt-1-1:numericValue "101936"^^xsd:double ;
  qudt-1-1:unit qudt-unit-1-1:Pascal ] ;
```

Aqui, estamos representando o resultado da observação. O valor observado é de 101936 Pascais.

```
sosa:resultTime [
  rdf:type time:Instant ;
  time:inXSDDateTimeStamp "2017-06-06T12:36:13+00:00"^^xsd:dateTimeStamp ] .
```

Por último, estamos indicando o momento exato em que a observação foi realizada, codificado no padrão ISO8601.

Dado que as triplas estão armazenadas em um servidor SPARQL, você pode acessá-lo, por exemplo, através de uma interface Tornado, que retorna uma página SPARQL. Em seguida, pode-se escrever uma consulta em SPARQL para recuperar informações específicas. Ao executar essa consulta, o servidor retornará uma lista de resultados correspondentes à consulta.

1.4 Semana 04

Aula 4: endpoint SPARQL

28 Agosto 2023

Nota 1. Com a biblioteca rdflib, não precisa explicitamente escrever strings no formato Turtle, porque a biblioteca oferece uma API mais abstrata e Pythonic para construir grafos RDF.

o WoT-py para criar um servidor que expõe "Things" conforme os princípios do Web of Things (WoT) do W3C. Este servidor WoT está construído em cima do framework Tornado para lidar com requisições HTTP.

O WoT-py é um framework específico para criar, expor e consumir "Things" em conformidade com WoT. É uma abstração para a criação de dispositivos IoT compatíveis com WoT.

O Tornado, por outro lado, é um framework web geral para Python. Ele pode lidar com requisições HTTP, WebSocket e outros. É um servidor web assíncrono e também oferece ferramentas para criar aplicativos web.

Para adicionar um endpoint SPARQL ao servidor, está essencialmente adicionando mais funcionalidade ao servidor Tornado subjacente que está sendo usado por WoT-py.

- WoT-py: Esta é a camada superior que está lidando com a exposição e comunicação dos "Things". Ele se encarrega de detalhes específicos de WoT.
- Tornado: Esta é a camada inferior que efetivamente ouve e responde às requisições HTTP. É o servidor web real.

1.5 Semana 05

Aula 5: endpoint SPARQL

18 Setembro 2023

implementação combinada do servidor WoTPy com o servidor SPARQL separado. A ideia é ter dois serviços rodando em paralelo: um para o WoT e outro para lidar com consultas SPARQL relacionadas aos dados semânticos dos dispositivos.

- Definição de Variáveis Globais e Portas:
  - HTTP_PORT e SPARQL_PORT definem as portas nas quais os servidores WoT e SPARQL serão executados, respectivamente.
  - uv_data é uma variável global que armazena os dados do sensor UV.
- Configuração do Grafo RDF:
  - O RDF é utilizado para representar dados em um formato padrão, tornando mais fácil o compartilhamento de dados entre diferentes domínios.

  - O código define diferentes namespaces e classes que serão usados para representar as informações do sensor UV de forma semântica.

- Descrição WoT:
  - A variável description define uma descrição WoT para um sensor UV, que tem uma propriedade chamada UV_SENSOR.
- SPARQLHandler:
  - Esta é uma classe que herda de tornado.web.RequestHandler. Ela é usada para lidar com solicitações HTTP POST no endpoint /sparql.
  - Quando recebe uma solicitação, ela lê a consulta SPARQL do corpo da solicitação, executa a consulta no grafo RDF e retorna os resultados em formato JSON.
- SPARQLServer:
  - Uma aplicação Tornado que define o servidor SPARQL. Ela apenas contém o endpoint /sparql, que é atendido pelo SPARQLHandler.
- CustomHTTPServer:
  - Esta classe estende o HTTPServer de WoTPy. Neste exemplo simplificado, não foram adicionadas funcionalidades extras além do que já está presente no HTTPServer.
- Métodos de Leitura e Escrita UV:
  - read_uv(): Lê os dados do sensor UV da variável global uv_data.
  - write_uv(): Atualiza a variável uv_data e modifica o grafo RDF para refletir os novos dados.
- Funções de Inicialização dos Servidores:
  - start_server(): Inicializa e começa a execução do servidor WoTPy.
  - start_sparql_server(): Inicializa e começa a execução do servidor SPARQL.
- Execução Principal:
  - Se o script for executado como o programa principal (não como um módulo importado), ele inicia os dois servidores em paralelo.

A combinação desses componentes permite que os dispositivos se comuniquem usando o padrão WoT enquanto os dados são representados semânticamente usando RDF. Além disso, o servidor SPARQL permite consultas complexas sobre esses dados semânticos. Por exemplo, é possível consultar todos os dispositivos que têm uma leitura UV acima de um certo valor.

1.5.1 Consultas SPARQL

Para fazer consultas SPARQL usando curl, será necessário enviará um pedido POST com a consulta SPARQL como corpo da solicitação. Supondo uma solicitação de todos os triplos (sujeito-predicado-objeto) no grafo RDF:

```
curl -X POST -H "Content-Type: text/plain" -d "SELECT * WHERE { ?s ?p ?o . }" http://loc
```

- X POST: Define o método HTTP como POST.
- H "Content-Type: text/plain": Define o cabeçalho "Content-Type" da solicitação. Dado o seu código anterior, espera-se que as consultas SPARQL sejam enviadas como texto simples (text/plain).
- d "SELECT * WHERE ?s ?p ?o . ": A consulta SPARQL. Ela pede por todos os sujeitos (?s), predicados (?p) e objetos (?o) no grafo. O * no SELECT significa que você quer todas as variáveis listadas nas cláusulas WHERE.
- http://localhost:8585/sparql: A URL do endpoint SPARQL.

1.6 Semana 06

Aula 6: Mecanismo de Persistência

25 Setembro 2023

O armazenamento persistente refere-se a qualquer forma de armazenamento de dados que mantém a integridade e a disponibilidade desses dados, mesmo após a interrupção ou reinicialização de um sistema. Em outras palavras, os dados persistidos não são perdidos quando a energia é desligada, quando o sistema é reiniciado ou quando a aplicação é encerrada.

A rdflib oferece vários mecanismos de persistência, permitindo armazenamento de grafos RDF em diversos backends de armazenamento, desde bancos de dados relacionais até sistemas de banco de dados NoSQL. O uso de persistência é especialmente útil quando se trabalha com grandes conjuntos de dados ou quando se quer persistir dados entre execuções de um programa.

Armazenamento em Memória (para conjuntos de dados menores):

```
from rdflib import Graph

g = Graph(store='IOMemory')
```

Berkeley DB:

É um software de banco de dados de alto desempenho utilizado para armazenamento chave-valor. A rdflib tem um plugin para Berkeley DB (via bsddb3), permitindo que os dados RDF sejam armazenados de forma persistente usando Berkeley DB.

SQLite:

É uma biblioteca C que fornece um banco de dados relacional leve. A rdflib também possui um plugin para SQLite, o que significa que você pode armazenar seu grafo RDF em um banco de dados SQLite.

OWLReady:

É uma biblioteca Python dedicada ao uso de ontologias OWL. A biblioteca permite carregar, modificar e salvar ontologias OWL. Por baixo dos panos, o OWLReady usa SQLite para armazenar dados de ontologia de forma persistente. Contudo, vale ressaltar que OWLReady não usa rdflib por padrão; ele tem sua própria maneira de lidar com ontologias e RDF.

Usando rdflib com Berkeley DB:

```
from rdflib import Graph

storepath = "path_to_directory_for_store"
g = Graph('Sleepycat', identifier='testgraph')
g.open(storepath, create=True)
```

Usando rdflib com SQLite:

```
from rdflib import Graph

storepath = "path_to_sqlite_database_file"
g = Graph('SQLite', identifier='testgraph')
g.open(storepath, create=True)
```

Usando OWLReady com SQLite:

```
from owlready2 import *

default_world.set_backend(filename="path_to_sqlite_database_file")
```

Integrar rdflib e OWLReady, precisará fazer um trabalho manual, como importar dados de rdflib para OWLReady ou vice-versa. No entanto, não há uma integração direta out-of-the-box entre rdflib e OWLReady, e precisaria cuidadosamente gerenciar as operações para evitar conflitos ou sobreposição de dados.

Berkeley DB pode ser mais rápido para certas operações e tem uma longa história de uso em ambientes de alto desempenho, enquanto SQLite oferece uma solução mais leve e portátil com um modelo relacional completo. Se usar o OWLReady, pode fazer sentido ficar com SQLite, já que é o backend padrão da OWLReady.

1.6.1 Armazenamento Persistente com Berkeley DB

O Berkeley DB (comumente abreviado como BDB) é uma biblioteca de software destinada a fornecer um mecanismo de armazenamento de banco de dados de alto desempenho para aplicativos. Originalmente desenvolvida pela Sleepycat Software, a BDB é agora de propriedade e mantida pela Oracle Corporation.

Modelo de Armazenamento de Chave-Valor: No seu núcleo, o Berkeley DB é um banco de dados orientado a chave-valor. Isso significa que ele armazena dados como pares de chave-valor, onde a chave é usada para recuperar rapidamente o valor associado.

Incorporável: Ao contrário de muitos sistemas de gerenciamento de banco de dados (DBMS) que são projetados para serem executados como serviços autônomos, o Berkeley DB é uma biblioteca que pode ser vinculada diretamente a um aplicativo. Isso o torna mais leve e, muitas vezes, mais rápido para certos usos do que um DBMS completo.

Transações ACID: O Berkeley DB suporta propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), o que o torna confiável para armazenar dados críticos.

Concorrência: Ele suporta acesso concorrente por múltiplos threads ou processos e utiliza bloqueios para manter a integridade dos dados.

Recuperação após Falhas: O BDB inclui suporte para recuperação após falhas, permitindo que os aplicativos se recuperem de falhas de hardware ou software sem perda de dados.

Tipos de Armazenamento: Além do básico armazenamento de chave-valor, o BDB oferece outros modelos, como Queues, Pilhas e Armazenamento baseado em Recursos.

Flexibilidade: O BDB não impõe um esquema específico aos dados e pode armazenar quase qualquer tipo de dados binários como valor, desde strings simples até objetos serializados ou blobs.

Integração com Linguagens e Sistemas: Berkeley DB oferece APIs para várias linguagens de programação, incluindo C, C++, Java, Perl, Python, entre outras. Ele também pode ser usado como um backend de armazenamento para sistemas maiores, como o OpenLDAP.

Licenciamento: O BDB é software proprietário. Ele foi originalmente lançado sob uma licença que permitia seu uso gratuito em aplicações de código aberto, mas essa política foi alterada depois que a Oracle adquiriu a Sleepycat. Portanto, é importante verificar o licenciamento atual se você estiver considerando usar o BDB em um produto ou projeto.

Uso: Dado o seu design leve e rápido, o Berkeley DB é comumente usado em sistemas embutidos, sistemas operacionais, aplicações de email e web browsers, entre outros.

Instalar a biblioteca do Berkeley DB:

No Dockerfile, instalar as bibliotecas e ferramentas do Berkeley DB.

```
RUN apt-get update && apt-get install -y libdb5.3-dev
```

Instalar a biblioteca Python para o Berkeley DB:

Será necessário o módulo bsddb3 para usar o Berkeley DB com Python. Adicionar este módulo ao requirements.txt ou instalar diretamente usando pip no Dockerfile.

```
pip install bsddb3
```

```
# Adicione os imports necessários no início do seu código
from rdflib.plugins.stores import Sleepycat

# ...

# Especifique o diretório onde o Berkeley DB armazenará os dados
STORE_DIR = "path_to_directory_for_berkeleydb_store"

# Substitua a criação do gráfico em memória pelo Berkeley DB
g = Graph('Sleepycat', identifier='mygraph')
g.open(STORE_DIR, create=True)

# ...

# Adicione uma função de encerramento para garantir que o Berkeley DB seja fechado corretame
def close_store():
    g.close()

if __name__ == "__main__":
    # ...

    # Certifique-se de chamar close_store antes de sair do programa
    try:
        IOLoop.current().start()
    finally:
        close_store()
```

Quando se trabalha com armazenamento persistente, erros podem ocorrer (por exemplo, problemas de permissão de escrita, o banco de dados está corrompido, etc.). Tratar possíveis erros com blocos try-except adequados. Além disso, é uma boa prática sempre fechar o banco de dados corretamente para garantir que todos os dados sejam gravados e que os recursos sejam liberados.

Consultar os dados usando o endpoint SPARQL:

```
curl -X POST -H "Content-Type: text/plain" -d "SELECT * WHERE { ?s ?p ?o . }" http://loc
```

1.6.2 Capacidade de serializar (ou exportar) dados RDF em diferentes formatos

Formato Turtle (TTL): print(g.serialize(format="turtle").decode("utf-8"))

Formato XML (RDF/XML): print(g.serialize(format="xml").decode("utf-8"))

Formato Notation3 (N3): print(g.serialize(format="n3").decode("utf-8"))

Formato JSON-LD: print(g.serialize(format="json-ld").decode("utf-8"))

Formato NTriples (NT): print(g.serialize(format="nt").decode("utf-8"))

```
with open("output.ttl", "wb") as file:
    file.write(g.serialize(format="turtle"))
```

Ao usar a função serialize, o formato desejado é especificado usando o argumento format. A saída da função é um objeto bytes, por isso é frequentemente decodificado para utf-8 para exibição ou armazenamento em formato de texto.

1.7 Semana 07

Aula 7: Interface Gráfica

02 Outubro 2023

Aplicações desktop e aplicações web são dois tipos distintos de softwares que se diferenciam principalmente pelo modo como são acessadas e executadas.

Aplicação Desktop

- Execução Local: Roda diretamente no sistema operacional do computador do usuário.
- Acesso Offline: Pode ser usada sem a necessidade de uma conexão com a internet.
- Recursos do Sistema: Tem acesso direto aos recursos do sistema operacional e hardware.
- Instalação: Normalmente requer instalação e, por vezes, configuração no computador do usuário.
- Interface Gráfica: Usa os widgets e elementos de interface gráfica do sistema operacional.
- Atualizações: Os usuários geralmente precisam atualizar o software manualmente ou através de um sistema de atualização embutido.
- Exemplo de Tecnologia: Tkinter, PyQt para Python.

Aplicação Web

- Execução Remota: Roda em um servidor web e é acessada por meio de um navegador de internet.

- Acesso Online: Geralmente, necessita de uma conexão com a internet para ser utilizada (a menos que seja projetada para funcionar offline).
- Recursos Limitados: As operações no lado do cliente têm acesso limitado aos recursos do sistema do usuário devido às restrições de segurança do navegador.
- Sem Instalação: Não requer instalação no computador do usuário, apenas um navegador web.
- Interface Gráfica: Usa tecnologias web (HTML, CSS, JavaScript) para a interface, sendo independente do sistema operacional.
- Atualizações: Atualizações são feitas no servidor, sem necessidade de intervenção do usuário para atualizar a aplicação.
- Exemplo de Tecnologia: Flask, Django, Tornado para Python.

1.7.1 Aplicação Web

No código, foi adicionado um novo endpoint e um método para renderizar uma página HTML simples:

```
# Importando módulo para renderização de templates do Tornado
from tornado.web import RequestHandler, Application, url

# ...

class SparqlQueryHandler(RequestHandler):
    def get(self):
        # Renderizando o template HTML no acesso GET
        self.render("sparql_query.html")

    def post(self):
        # Obtendo a consulta SPARQL do formulário
        sparql_query = self.get_body_argument("sparql_query")

        # Executando a consulta
        results = g.query(sparql_query)

        # Enviando os resultados para serem exibidos no HTML
        self.render("sparql_query.html", results=results.serialize(format="json"))

class SPARQLServer(Application):
    def __init__(self):
        handlers = [
            ("/sparql", SPARQLHandler),
            ("/query", SparqlQueryHandler)  # Adicionando novo endpoint
        ]
```

```
        super(SPARQLServer, self).__init__(handlers)
```

E foi criado um arquivo sparql_query.html com um formulário HTML básico para input da consulta SPARQL e exibição dos resultados.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SPARQL Query Interface</title>
</head>
<body>
    <h1>Execute SPARQL Query</h1>
    <form method="post" action="/query">
        <label for="sparql_query">SPARQL Query:</label><br>
        <textarea name="sparql_query" id="sparql_query" rows="4" cols="50"></textarea><br>
        <input type="submit" value="Execute Query">
    </form>

    {% if results %}
    <h2>Results:</h2>
    <pre>{{ results }}</pre>
    {% end %}
</body>
</html>
```

1.7.2 Exemplo

Para o exemplo, foi colocado no SPARQL Query o seguinte comando: "SELECT * WHERE ?s ?p ?o . "

1.8 Semana 08

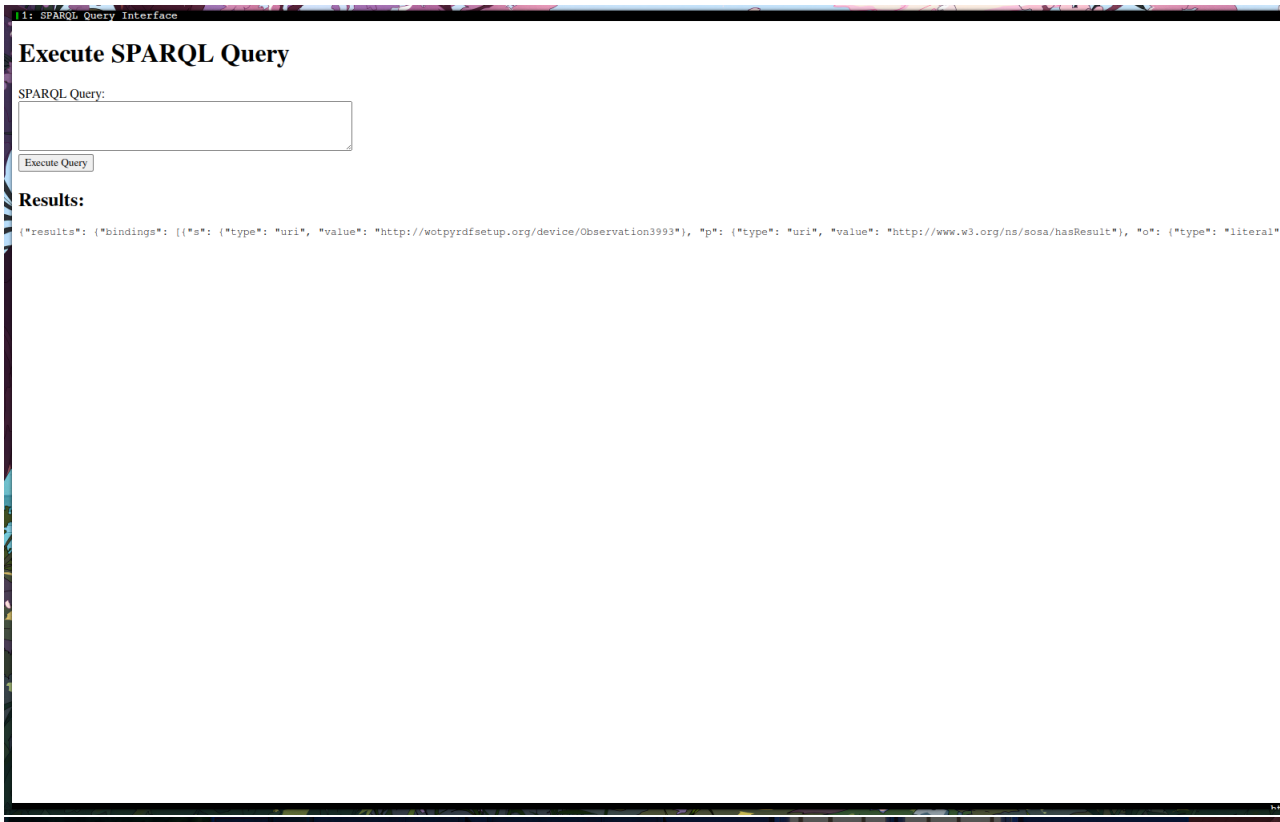
Aula 8: Memória de Persitência

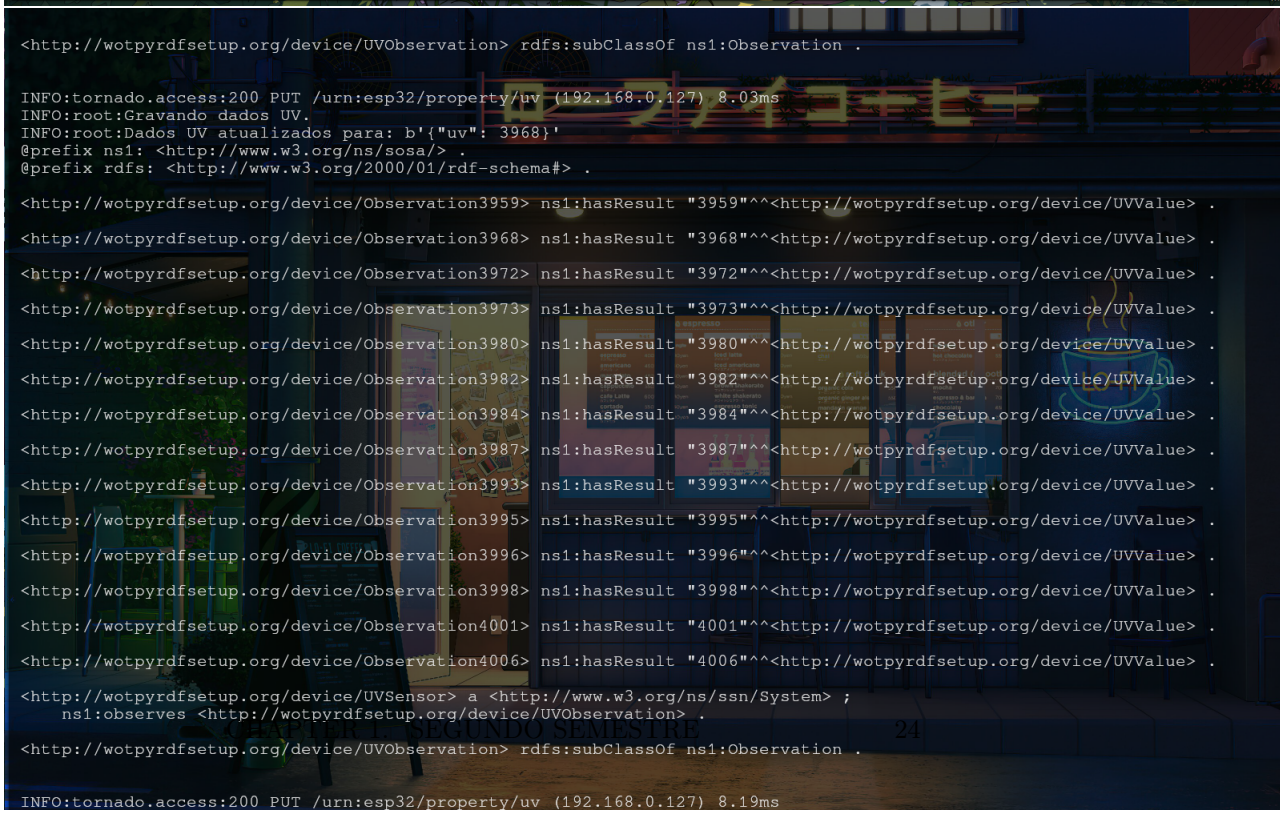
24 Outubro 2023

Para usar um banco de dados como memória de persistência, será utilizado o SQLite para um começo simples. Ele é um banco de dados relacional que armazena tudo em um único arquivo e não requer um servidor separado.

Para armazenar as observações em um banco de dados SQLite e, posteriormente, realizar consultas SPARQL a partir dessas observações, serão seguido os seguintes passos:

- Inicializar o SQLite Database: Ao iniciar o programa, criar um banco de dados SQLite (se ele ainda não existir) e uma tabela para armazenar as observações UV.


1: SPARQL Query Interface
Execute SPARQL Query
SPARQL Query:
Execute Query
Results:
{"results": {"bindings": [{"s": {"type": "uri", "value": "http://wotpyrdfsetup.org/device/Observation3993"}, "p": {"type": "uri", "value": "http://www.w3.org/ns/sosa/hasResult"}, "o": {"type": "literal"


<http://wotpyrdfsetup.org/device/UVObservation> rdfs:subClassOf ns1:Observation .
INFO:tornado.access:200 PUT /urn:esp32/property/uv (192.168.0.127) 8.03ms
INFO:root:Gravando dados UV.
INFO:root:Dados UV atualizados para: b'{"uv": 3968}'
@prefix ns1: <http://www.w3.org/ns/sosa/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<http://wotpyrdfsetup.org/device/Observation3959> ns1:hasResult "3959"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation3968> ns1:hasResult "3968"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation3972> ns1:hasResult "3972"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation3973> ns1:hasResult "3973"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation3980> ns1:hasResult "3980"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation3982> ns1:hasResult "3982"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation3984> ns1:hasResult "3984"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation3987> ns1:hasResult "3987"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation3993> ns1:hasResult "3993"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation3995> ns1:hasResult "3995"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation3996> ns1:hasResult "3996"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation3998> ns1:hasResult "3998"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation4001> ns1:hasResult "4001"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/Observation4006> ns1:hasResult "4006"^^<http://wotpyrdfsetup.org/device/UVValue> .
<http://wotpyrdfsetup.org/device/UVSensor> a <http://www.w3.org/ns/ssn/System> ;
ns1:observes <http://wotpyrdfsetup.org/device/UVObservation> .
<http://wotpyrdfsetup.org/device/UVObservation> rdfs:subClassOf ns1:Observation .
INFO:tornado.access:200 PUT /urn:esp32/property/uv (192.168.0.127) 8.19ms

- Adaptar a Função write_uv: Além de adicionar a observação ao grafo RDF, também insirir a observação no banco de dados SQLite.
- Sincronizar o Grafo RDF com o SQLite: Antes de executar consultas SPARQL, carregar os dados do SQLite no grafo RDF para garantir que ele esteja atualizado.

1.8.1 SQLite

SQLite é um mecanismo de banco de dados SQL embutido, que não requer um processo de servidor dedicado. Ele permite armazenar dados em um arquivo local e é uma opção popular para aplicações que necessitam de um banco de dados leve, como aplicações de desktop ou dispositivos móveis.

1. Inicialização do Banco de Dados:
   - DATABASE_NAME = 'observations.db': Define o nome do arquivo onde o banco de dados SQLite será armazenado.
   - init_db(): Esta função é responsável por inicializar o banco de dados. Se a tabela uv_observations não existir, ela será criada.
2. Adicionando Dados ao Banco de Dados:
   - add_to_db(value): Esta função é usada para adicionar uma observação UV ao banco de dados.
3. Lendo Dados do Banco de Dados e Adicionando ao Gráfico RDF:
   - load_data_to_graph(): Esta função lê as observações UV do banco de dados SQLite e as adiciona ao gráfico RDF.

Quando o script é iniciado (usando if __name__ == "__main__":), ele começa inicializando o banco de dados com a função init_db(). Ao receber dados UV, o servidor chama a função write_uv(value), que processa os dados e os adiciona ao banco de dados usando a função add_to_db(uv_val). Quando um cliente faz uma requisição SPARQL para buscar dados, a função load_data_to_graph() é chamada para carregar os dados UV do SQLite e adicioná-los ao gráfico RDF. Em seguida, a consulta SPARQL é executada no gráfico RDF.

1.9 Semana 09

Aula 9: Memória de Persitência

05 Novembro 2023

É importante notar que o método g.remove((UVSensor, Observes, None)) está sendo chamado, mas isso não significa necessariamente que todos os triplos existentes relacionados a observações anteriores estão sendo removidos. O método remove() está sendo aplicado de forma específica ao triplo que liga UVSensor com qualquer objeto através da propriedade Observes.

Este comportamento sugere que a intenção é assegurar que o sensor UV (UVSensor) esteja sempre ligado à última observação. No entanto, um sensor pode observar múltiplas instâncias de observações, e cada uma dessas observações pode ter seu próprio resultado. Normalmente, não se removeria observações anteriores a menos que você desejasse manter apenas a mais recente por algum motivo específico, o que não é comum em cenários de coleta de dados de sensores onde você deseja manter um histórico.

A remoção dos triplos existentes antes de adicionar novos pode ser um erro no design do sistema, ou pode ser intencional para um caso de uso específico não explicado no código fornecido. Se o objetivo é ter um histórico de observações, então essa linha deveria ser removida para permitir que o grafo acumule triplos de todas as observações sem sobreposição.

Para manter o histórico de observações sem duplicar informações, poderia ser usado um identificador único para cada observação. Esse identificador pode ser uma combinação de data e hora da observação, um número sequencial ou qualquer outra forma de chave única que faça sentido no contexto da sua aplicação.

Usando essa abordagem, cada observação terá um URI único e você pode manter todas as observações sem se preocupar com duplicatas. O método datetime.utcnow().strftime("%Y%m%d%H%M%S%f") gera uma string de data e hora em UTC que será quase certamente única para cada chamada, especialmente se as observações não forem geradas mais de uma vez por milissegundo, o que é o caso na maioria das aplicações práticas.

1.10 Semana 10

Aula 10: Memória de Persitência

19 Novembro 2023

1.10.1 Mapear o Volume para um Diretório no Host

Quando inicia o contêiner, mapear um diretório no host para o volume dentro do contêiner. No comando docker run, usar a flag -v ou –mount. Por exemplo:

```
docker run -v /path/on/host:/path/in/container myimage
```

Neste exemplo, /path/on/host é o caminho no sistema de arquivos do host e /path/in/container é o caminho dentro do contêiner onde você deseja que os dados sejam armazenados.

Os dados escritos pelo aplicativo no diretório /path/in/container dentro do contêiner serão armazenados no diretório /path/on/host no sistema de arquivos do host. Isso significa que mesmo se o contêiner for destruído, os dados ainda estarão disponíveis no host.

Mapear o diretório examples/uv$_sensordoseusistemahostdiretamentepara/appdentrodocontiner$.

Como /app é o diretório de trabalho padrão no contêiner, quando você executa ls, ele lista os conteúdos do diretório mapeado, que é o mesmo que o diretório de

trabalho. Por isso, você vê os arquivos de examples/uv$_s$*ensordoseusistemahost*.

Para evitar problemas de permissão, configurar o Docker para executar seus contêineres como um usuário não-root. Isso pode ser feito especificando um usuário no Dockerfile ou no comando docker run com a opção –user.

```
docker container run --network host -it --rm --user t1k -v $PWD/examples/uv_sensor:/app
```