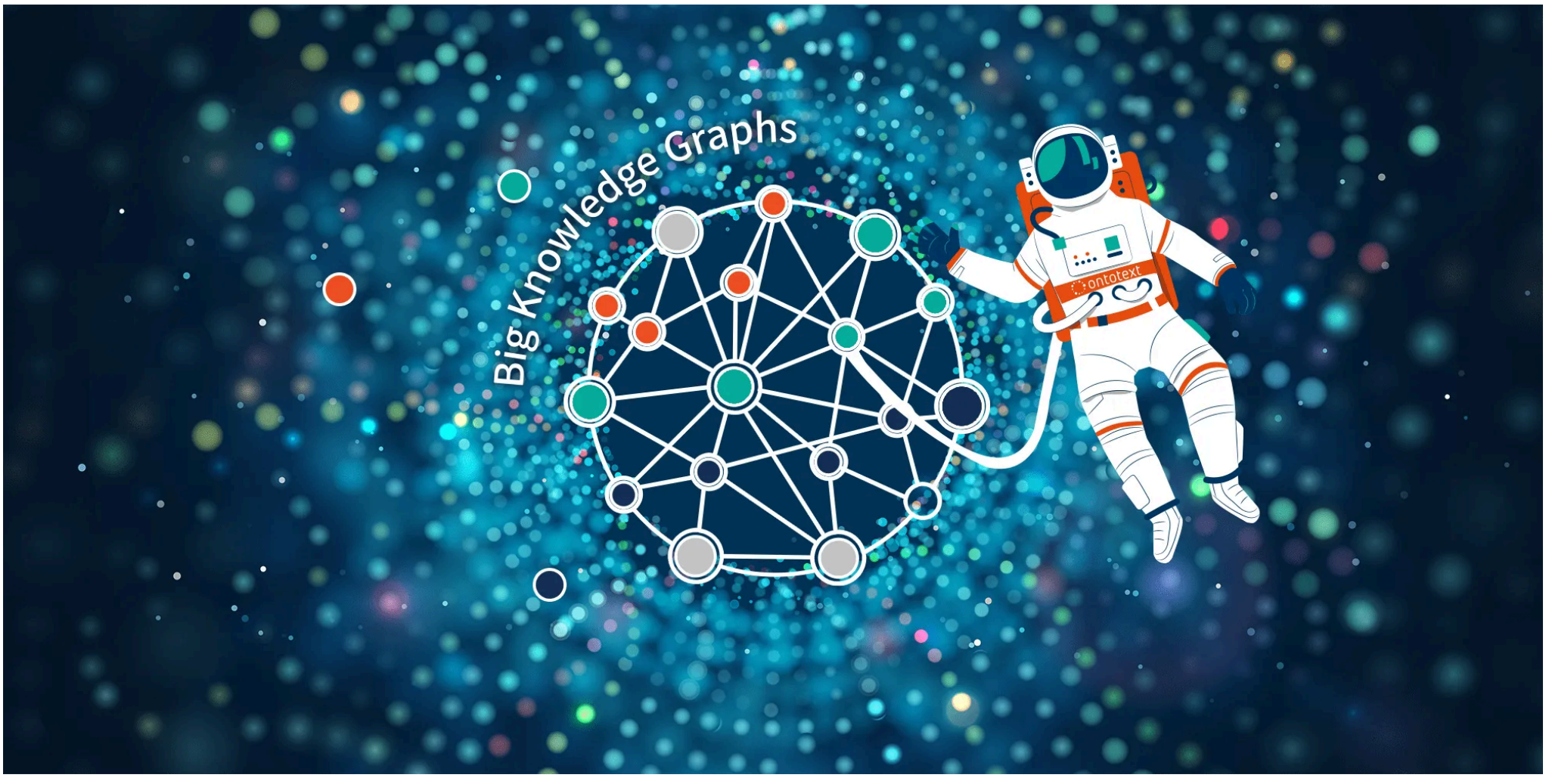
Blog > Technology

Reasoning with Big Knowledge Graphs: Choices, Pitfalls and Proven Recipes

How to use reasoning to enrich big knowledge graphs with new facts and relationships, avoiding the typical pitfalls and reaping all the benefits

July 8, 2022 12 mins. read  Atanas Kiryakov



Ontotext has been doing knowledge graphs way before the term became popular. Their main advantages are that they allow us to link data across different datasets and to use semantic schema and semantic metadata to describe this data, so that both humans and machines can interpret it.

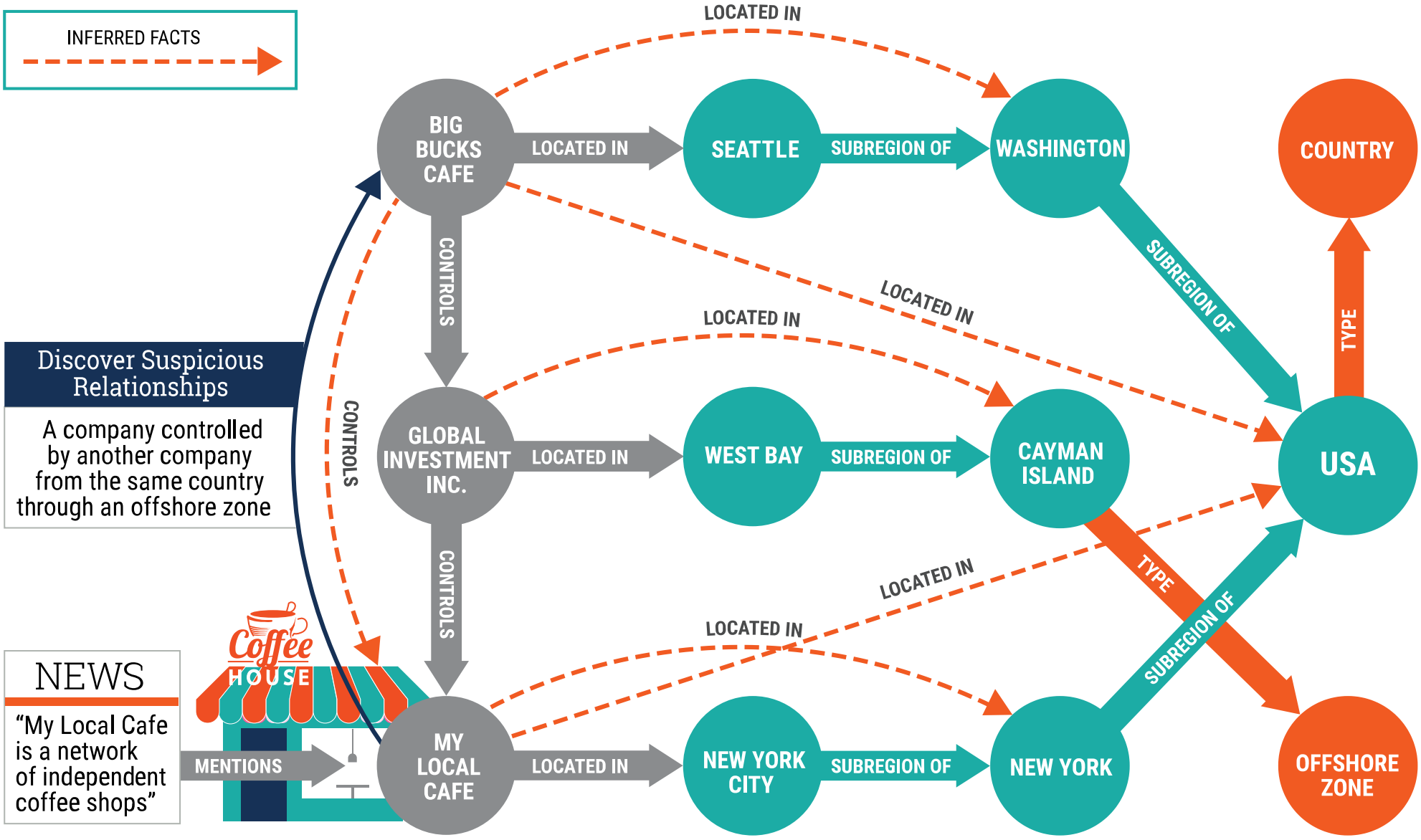
Essentially, knowledge graphs can be seen as a combination of three well-established paradigms in computer science: as a database that can be queried, as a big network structure that can be analyzed and as a knowledge base where new facts can be inferred.

Read on to learn more about the latter!

Introduction to Reasoning

Now, let's consider the following example sentence: "My Local Cafe is a network of independent coffee shops", illustrated on the diagram below.

Through inference, a knowledge graph can provide additional information that is not explicitly stated (follow the dashed orange arrows!). This can be information about companies, i.e., that Big Bucks Cafe controls Global Investment Inc., which controls My Local Cafe. Or about geographic location, i.e., that Big Bucks Cafe is located in Seattle, Global Investment Inc. is in West Bay and my Local Cafe is in New York. This

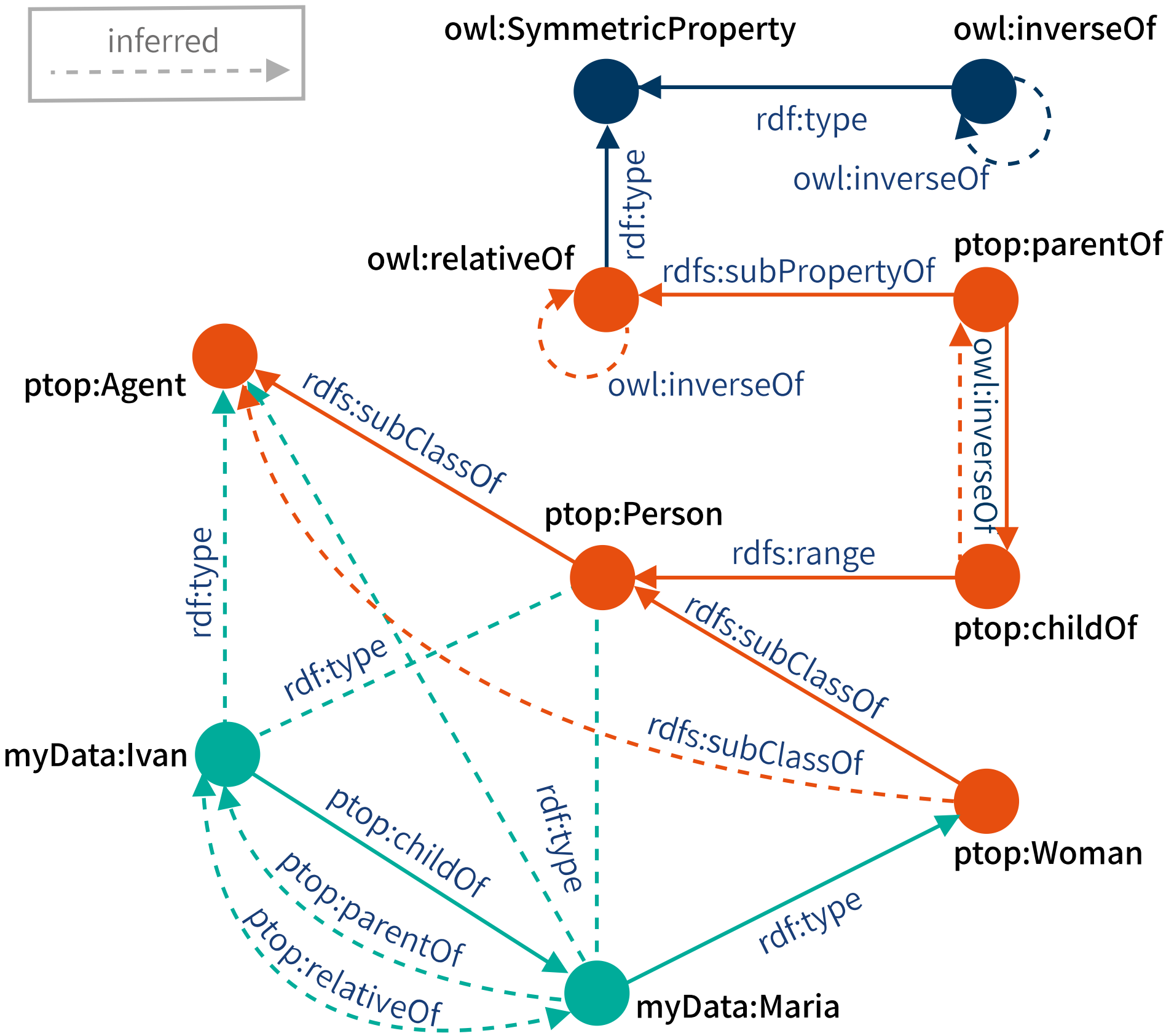


information can be further supplemented by the facts that Seattle is in the state of Washington and New York is in the state of New York, both of which are part of the U.S., while West Bay is a district in Grand Cayman – one of the the Cayman Islands.

Given the right semantic schema (known as ontology) behind it, a graph database engine can infer additional facts and relationships, which can then be added to the knowledge graph. This enables you to discover multi-hop relationships or other patterns that might otherwise be difficult to catch. If, for example, you want to know which companies in the U.S. control another company in the U.S. through a company in an offshore zone, many databases can work it out for you. But a graph database with inference, like Ontotext's GraphDB, makes it much easier and faster to put the data together and execute such queries.

Enter Semantics

Graph databases, based on the RDF model, deal with both the semantic schema and the instance data as a single graph structure so that queries can span across schema, data and metadata. In formal semantics, which uses Semantic Web standards like RDF Schema and OWL, the semantic schema defines the meaning of the different classes and types of relationships. The instance data on the other hand maintains information specific to individual instances – typically entities like people, organizations and locations or abstract topics and concepts like the number Pi, the musical style Ska and The Istanbul Convention.



The example above shows the schema part of the graph in green – a class hierarchy (*Woman* is a subclass of *Person*, which is a subclass of *Agent*) and some definitions of relationships (*childOf*, *parentOf* and *relativeOf*). And in orange is the specific instance data like the fact that *Ivan* is *childOf Maria*. There are also a couple of system primitives used to define the semantics of property definitions in blue: SymmetricProperty and InverseOf.

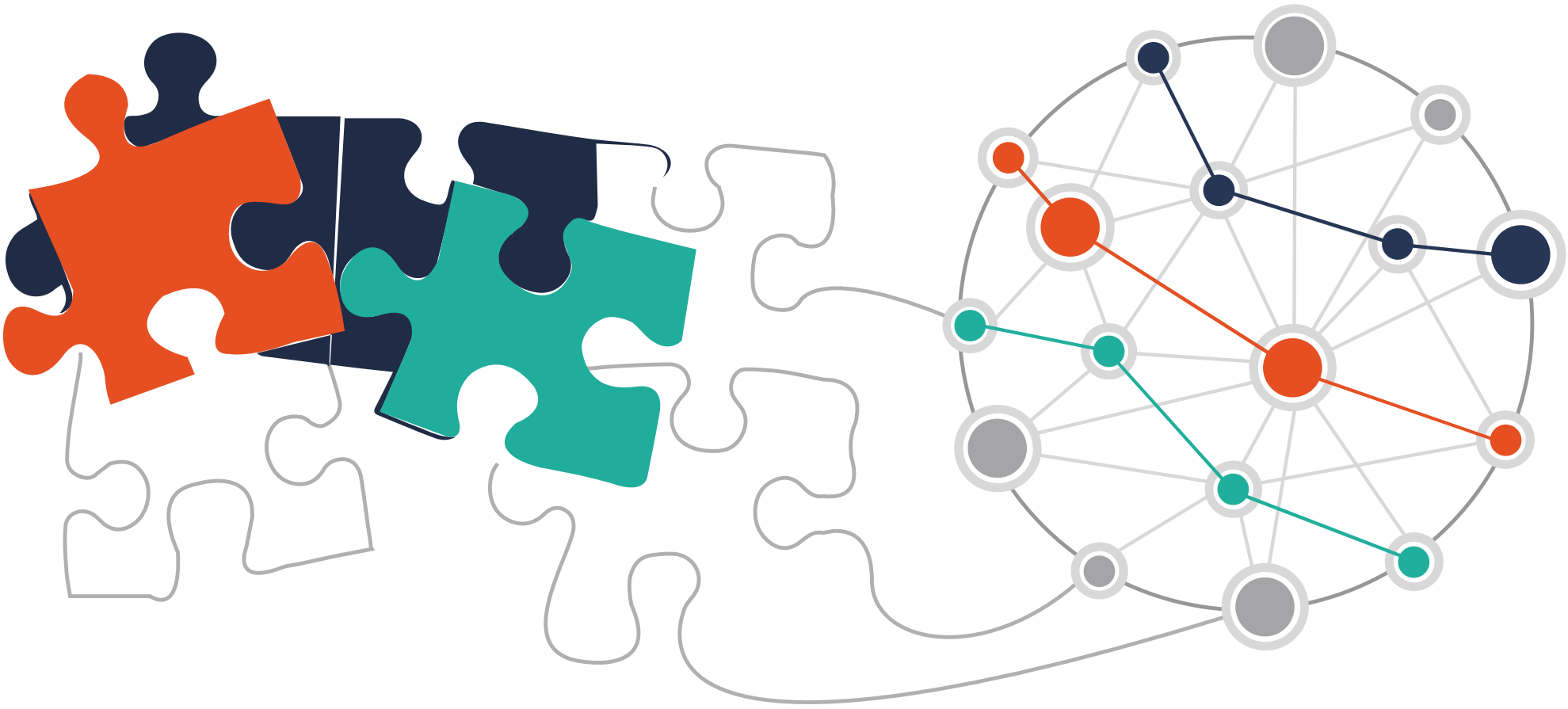
Based on the explicit facts, a graph database engine can automatically infer other facts and relationships (illustrated again by the dashed line arrows). For example, if *Woman* is *subclassOf Person* and *Person* is *subclassOf Agent*, the engine will infer that *Woman* is *subclassOf Agent*, because *subclassOf* is a transitive property. Going further, if *Maria* is an instance of *Woman*, then *Maria* is also an instance of *Person* and *Agent*.

So far, we have encoded some very basic object-oriented semantics using logic and rules. But we can go even further. If *Ivan* is *childOf Maria* and the schema describes this relationship as the reverse of *parentOf*, then it can be inferred that *Maria* is *parentOf Ivan*. And because *parentOf* is a special case of *relativeOf*, which is defined as a symmetric property, then *Maria* is *relativeOf Ivan* and *Ivan* is *relativeOf Maria*.

Our example is intentionally simple because, when you define the semantics of your schema, you should do the minimum that gets the job done. As everybody knows, being a parent or a child of someone involves a lot more than what we have provided in the schema. In the real world, there are plenty of other things we may want to say about these relationships other than that they are inverse to one another and a special case of relatives. But in ontologies you want to keep things as simple as possible and define only the semantics that is relevant to your application.

Why Go To All This Trouble?

When you have data coming from multiple sources, before you can start doing anything with it, you need to align the different data modeling patterns. This is necessary because one dataset may use the *parentOf* relationship to link parent to child, another – *childOf* and a third – just the more general *relativeOf*.



But after you've aligned the different relationship types coming from the different schemas, you can have a query like: Give me all Xs that match the pattern "Maria relativeOf X" and it will return *Ivan* although the explicit relationship that comes from the source uses a more specific relationship going in the opposite direction. This very trivial example shows that even with such simple semantics and a couple of rules, it's easier to query data coming from multiple sources regardless of the different modeling patterns they use.

Another important thing about inference is that it makes it easy to detect patterns and get deeper insights. Most database engines don't do recursive investigations at query-time as it slows them down considerably. However, with reasoning, you can trace multi-hop relationships and at the same time have an efficient database engine, able to do a query with 10 joins and still get results in milliseconds.

And, last but not least, you can use inference to do consistency checking and quality validation to make sure that your data is correct before you start working with it.

Reasoning Pitfalls Or The Small Print

But despite all the advantages of using reasoning, when you start to experiment on a big scale, to get the engine to cope with inference without troublesome performance penalties involves a steep learning curve.

> There is a fine balance between complexity and practicality you have to capture in an ontology. If it's too small and simple, you can't infer many things from it and if it's too complex, inference becomes slow and unpredictable.
>
> SHARE ON X

However, from what we've seen again and again, projects using reasoning fail most often because people make modeling mistakes. One such typical mistake is that people over-engineer their ontologies by making the ontology language too expressive. This leads to inference results that are hard to understand and verify as well as poor performance.

Another frequent mistake is getting inappropriate reasoning support so the inference implementations works well with taxonomies and conceptual models of a few thousands concepts, but cannot cope with a knowledge graph of millions of entities. And one final thing we've seen a lot is that people expect reasoning with virtual knowledge graphs to work well, while the truth is that it's often infeasible.

Reasoning Implementation Choices

When it comes to reasoning, there are two ways in which it can be implemented in a database: forward-chaining or backward-chaining.

Forward-chaining starts with the data, infers everything that can be inferred and then stores it in the database. This means that every time data is loaded or updated, you have to do reasoning again. The technical term for making all possible inferences upon updates and storing them in the database is materialization. There are some disadvantages to forward-chaining and materialization, but if the modeling is done with consideration for performance and extensibility, it scales quite well.

The alternative to forward-chaining is backward-chaining, which allows more complex reasoning. In this case, reasoning is performed at query-time, so when you load or update data, no reasoning takes place. The drawback is that this approach spoils query optimization, which causes massive difficulties with query evaluation on big datasets.

To deliver decent query performance, when dealing with complex structured constraints and relationships (as in SQL, SPARQL and Cypher), engines must perform query optimization. It involves reordering of the so called joins, based on estimates about the (conditional) cardinalities of the anticipated results of the separate constraints. These estimates are derived from the database indices. When backward-chaining is used, reliable cardinality guesses are not available and query optimization is practically impossible. This means that only very rudimentary query processing is feasible, close to what people do in MS Excel and far away from the analytical capabilities of mature database engines.

Reasoning in GraphDB

In GraphDB, we use forward-chaining and materialization, which allows us to do efficient query evaluation on big datasets, even in cases when there are very high update rates. A unique capability of GraphDB is that reasoning works incrementally in both directions. When you insert new data, instant reasoning takes place to infer whatever new facts can be inferred and interferences that are no longer supported upon delete are retracted.

Reasoning works via entailment and consistency rules and in GraphDB there are a number of built-in rule-sets. They support specifications like RDFS, OWL 2 RL, OWL 2 QL as well as specific rule-sets, which we find useful like, for example, OWL-Horst. You can also define custom rule-sets, starting with an existing one and then removing the rules that you don't need for your application. Overall, the OWL 2 ontology and schema definition language is very powerful regarding class reasoning, i.e., inferring types, but surprisingly limited regarding inferring new properties

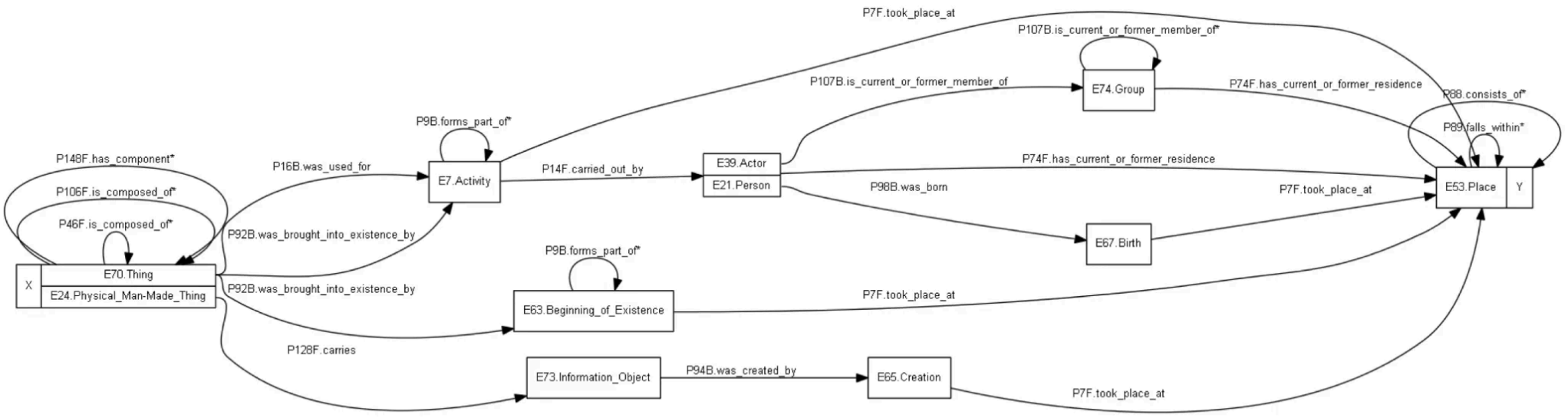
GraphDB also provides perfomance-optimized versions of all predefined rule-sets. These versions exclude some RDFS reasoning rules that are not useful for most applications, but add substantial reasoning overheads. One example is the rule that explicitly entails that every identifiable node in the graph is an instance of the class *rdfs:Resource*. It's up to you if you want to use them to optimize your performance or you want to stay with the fully standard compliant non-optimized versions.

Reasoning in Action

We have many success stories about reasoning with big knowledge graphs, but before we wrap up, let's look at just a few.

Search in the British Museum's Collection

We helped the British Museum model the artifacts in their collection with a very detailed ontology called CIDOC CRM. The semantics of this ontology made it easy to generalize the great variety of fundamental relationships. For example, the many different ways in which an artifact could be related to a geographical location (place of creation, place of discovery, birthplace of the artist who created it, etc.).



Click to enlarge

Back in 2012 when we worked on this project, the British Museum had something like 2 million artifacts described with about 200 million explicit statements in the database, i.e. about 100 statements per artifact. And when we ran the inference on them, they expanded four times and the total size of the repository became close to 1 billion statements.

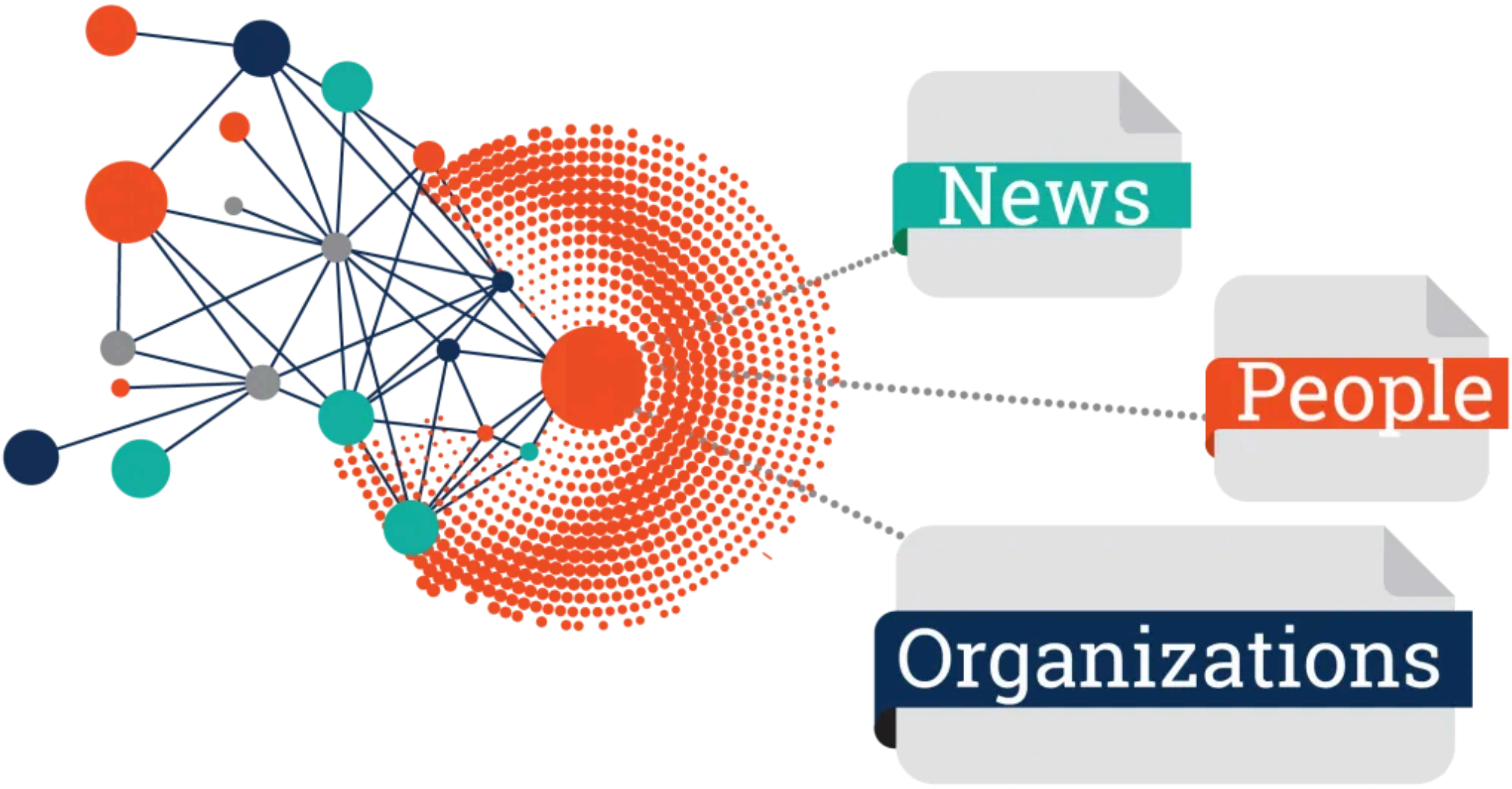
The BBC & the Semantic Publishing Benchmark

Another interesting project we worked on was with the BBC when they invented their Dynamics Semantic Publishing concept in 2010. The company wanted to use ontologies and an enterprise-ready graph database with the power of inference in order to minimize the expensive editorial management of their content assets. All content such as texts, pictures and videos was described with rich metadata to enable the generation of thematic web pages about specific players, teams, events, etc. The pilot project was BBC's website for the FIFA Worldcup in 2010.

This new approach to publishing and managing content provided the basis for one of the Linked Data Benchmarking Council's benchmarks – the Semantic Publishing Benchmark. This is an interesting example because it's a live real application where reasoning is applied in a very challenging database workload scenario, involving massive query loads in parallel with very frequent update transactions on top of a graph of millions of nodes and edges.

FactForge

One final example of reasoning with big knowledge graphs is our demonstrator FactForge. It's a hub of Linked Open Data datasets like DBPedia, GeoNames and some other smaller ones, and news metadata. We process about 2,000 articles a day and we perform Name Entity Recognition on top of them. All this metadata is stored in the database, amounting to about 2.2 billion explicit statements. And with inference on top of them, we add another 328 million inferred statements.



Wrap Up

Semantic data schemes and reasoning can make a difference in numerous cases. One such example is aligning vocabularies and modeling patterns across different data sources to enable easy querying of the data as if it all complies with a uniform data model. Another great example is exploring complex graph patterns and multi-hop relationships.

There are also plenty of lessons learned about the pitfalls of reasoning, which is by nature computationally complex. Designing ontologies that allow for querying big datasets goes much easier when done by an enterprise data architect or a DBA with some training in RDFS and OWL rather than a logician who feels engaged to employ all the capabilities of Description Logic.

Ontotext has implemented and optimized the inference in GraphDB based on 20 years of experience in reasoning over very big volumes of data. Our team is ready to accelerate your knowledge graph development and consult you on the relevant inference challenges.

Want to enrich your knowledge graph with new facts and relationships?

Give GraphDB a try today!

Download Now

Article's content

Related Posts

How We Do Text Analysis with Knowledge Graphs at Ontotext

Read about the special relationship between knowledge graphs & text analysis and the business value of delivering such solutions

Ontotext’s LinkedLifeData Inventory Or How to Get Value from Your Knowledge Graph in No Time

Read about Ontotext's methodology for building knowledge graphs coupled with an extensive experience working in Life Sciences, Pharma

and Healthcare.

5 Questions and Answers About Knowledge Graphs in Financial Services

Read about some common questions prospects ask about the differentiating capabilities of knowledge graphs as enablers in the Financial Services sector.

About Posts

Atanas Kiryakov

CEO at Ontotext

Atanas is a leading expert in semantic databases, author of multiple signature industry publications, including chapters from the widely acclaimed Handbook of Semantic Web Technologies.

ontotext

North America

Ontotext USA, Inc.
Select Office Suites Flatiron
1115 Broadway, 16 Madison Square West
Suite 1200, New York, NY 10010, USA
+1 929 239 0659

Europe

Twins Centre
fl.3, 79 Nikola Gabrovski str.
1700 Sofia, Bulgaria
+359 2 974 61 60

Switzerland Innovation Park
Basel Area
Hegenheimermattweg 167A
4123 Allschwil,
Switzerland

Get the latest headlines with our Newsletter:

First name

Last name

Email

I confirm that I have read and agree to the terms of **Ontotext's Privacy Policy** and consent to the processing of my personal data as described therein.*

Submit