Exploring Digital Twins integration with ASP and ASP Chef

Paola Guarasci[1]

[1]University of Calabria Rende(CS), Italy

Abstract

This research explores the integration of the Digital Twins Definition Language (DTDL) with ASP Chef, a web-based platform for Answer Set Programming (ASP). The goal of the research is to take advantage of the capabilities of ASP Chef to query, analyze, and visualize digital twins. As a first step in this direction, we introduce a method for mapping DTDL-defined digital twin models into ASP facts. This approach enables a seamless transition from high-level digital twin specifications to declarative reasoning and interactive visualization, offering a flexible framework for interpreting and exploring digital twin configurations.

Keywords

Asp, Digital twin, AspChef, DTDL

1. Introduction and problem description

Digital twins are virtual representations of physical entities, such as sensors, rooms, vehicles, or more complex systems, designed to reflect their structure, behavior, and real-time state. These digital counterparts enable simulation, monitoring, and data-driven reasoning over physical environments. The modeling of digital twins requires a precise and expressive language to describe their properties, telemetry, relationships, and component structure. To this end, Microsoft has developed the Digital Twins Definition Language (DTDL), a domain-specific modeling language tailored to define digital twin interfaces and instances. DTDL is built upon JSON-LD (JavaScript Object Notation for Linked Data), a serialization format for linked data that extends standard JSON. JSON-LD was introduced as a way to integrate structured data into the evolving Semantic Web using familiar JSON syntax, while maintaining compatibility with RDF (Resource Description Framework). RDF defines a data model to represent relationships between entities on the Web. JSON-LD serves as a concrete serialization of RDF data and in fact can be simultaneously interpreted both as valid JSON and as an RDF document. This foundation allows DTDL to support semantic interoperability, reuse, and extensibility through the definition of domain-specific ontologies. Models in DTDL can inherit from one another, enabling abstraction and specialization, and can either define custom concepts or reuse industry-specific vocabularies. For instance, the RealEstateCore ontology for smart buildings (https://github.com/Azure/opendigitaltwins-building) and the EnergyGrid ontology for energy systems (https://github.com/Azure/opendigitaltwins-building) are both provided as reference models within the Azure Digital Twins ecosystem. Technically, a DTDL model is defined as a forest of DTDL elements—interfaces, properties, telemetry, commands, and relationships—organized under unique identifiers called DTMIs (Digital Twins Model Identifiers). These models can be composed, extended, and reused across different domains and applications (https://github.com/Azure/opendigitaltwins-dtdl/blob/master/DTDL/v4/DTDL.Specification.v4.md).

ASP Chef [1, 2] is a web-based platform designed to support interactive development, execution, and visualization of logic programs within the paradigm of Answer Set Programming (ASP). While many ASP environments focus primarily on the writing and debugging of code, ASP Chef adopts a higher-level perspective, enabling users to design and execute pipelines of operations, known as recipes, that process and transform answer sets in a structured and modular fashion. Each recipe in ASP Chef is composed of a series of ingredients, where each ingredient encapsulates a specific operation: grounding

ICLP DC 2025: 21st Doctoral Consortium on Logic Programming, September 2025, Rende, Italy.

paola.guarasci@unical.it (P. Guarasci)

0009-0005-8763-1431 (P. Guarasci)

a logic program, solving it to compute answer sets, filtering results based on user-defined conditions, applying queries, generating new interpretations, or rendering output in visual form. This modular design allows users to construct complex logic-based workflows in a step-by-step manner, facilitating experimentation, compositional reasoning, and reuse of processing logic across different applications.

What sets ASP Chef apart is its ability to manage interpretation streams (i.e., sequences of partial or complete answer sets) through these pipelines. Rather than returning a static set of results, recipes can transform intermediate results, chain operations, and ultimately produce outputs that are either textual (e.g., ASP facts) or graphical (e.g., network visualizations). The visual layer is powered by templating systems such as Mustache, integrated with JavaScript visualization libraries like *@vis.js/Network*, enabling interactive graphical representations of logical models. The user interface of ASP Chef is entirely browser-based and designed for accessibility and ease of use, lowering the barrier for both newcomers and experts. Users do not need to install any external solvers or tools locally; the platform integrates standard ASP solvers such as CLINGO and handles grounding, solving, and visualization client-side. This makes it particularly suitable for educational contexts, rapid prototyping, and data-rich domains where logical models must be both computed and interpreted by humans.

The objective of this work is to enable the analysis and visualization of digital twins defined using DTDL within the declarative and visual framework offered by ASP Chef. While DTDL provides a standardized, extensible way to model the structure and semantics of digital twins, defining interfaces, properties, relationships, and telemetry, its focus remains on representation rather than reasoning. By translating DTDL models into ASP facts, we can take advantage of the expressive power of ASP to perform complex queries, detect inconsistencies, infer implicit knowledge, and simulate behaviors over digital twin models. Integrating these two technologies allows for the creation of flexible reasoning pipelines where DTDL-defined twins serve as the data layer, and ASP Chef recipes provide the logic and visualization layers. In this approach, the declarative logic encoded in ASP can be used to filter or analyze digital twin instances. At the same time, the built-in support for templating and graph-based visualization in ASP Chef makes it possible to render digital twin networks in an interactive and human-friendly form. This integration thus bridges high-level semantic modeling with logic-based analysis and visual feedback, empowering users to inspect, reason over, and communicate digital twin configurations more effectively. This integration is particularly relevant in contexts where explainability, rule-based control, or constraint checking over digital twins is essential. For example, in smart building scenarios, one may use DTDL to describe a the layout of a building, sensors, and control systems, and then use ASP to verify that sensor placement satisfies coverage constraints, or to detect redundant or missing components. In energy grids or manufacturing processes, logical rules may encode safety policies, energy efficiency conditions, or fault detection mechanisms that can be automatically applied to the representation of the digital twin.

2. Background and overview of the existing literature

Digital twins have gained significant attention in recent years as a paradigm for modeling, monitoring, and controlling cyber-physical systems. DTDL, developed by Microsoft, has emerged as a standard for describing digital twin interfaces in a structured and extensible way. It is widely used in industrial IoT settings, particularly within the Azure Digital Twins platform. While DTDL focuses on the semantic modeling of digital entities, it does not offer built-in mechanisms for logical reasoning or constraint checking. In contrast, ASP provides a formalism well-suited for representing complex knowledge and deriving conclusions under non-monotonic reasoning. ASP has been successfully applied in configuration [3, 4, 5], diagnosis [6, 7, 8], and planning [9, 10, 11], but its integration with digital twin technologies remains underexplored.

On the visualization side, tools such as vis.js have been used extensively for interactive graph visualization. ASP Chef [1] introduced its first visualization functionality in [2] through the addition of the *Graph* ingredient, a component designed to generate graph-based visualizations as side effects of logic program executions. This feature enabled users to embed visual directives within ASP programs

themselves, producing interactive graphical representations of answer sets. The approach follows the tradition of tools like ASPViz[12], IDPD3[13], and Kara [14], which link answer set semantics to visual metaphors. Building upon this foundation, recent versions of ASP Chef have taken a different route to enhance expressiveness and flexibility: rather than embedding visuals directly in ASP rules, they take advantage of Mustache templates to configure external JavaScript visualization libraries by querying answer sets. This model-driven approach allows ASP facts and their relationships to drive the generation of structured visual outputs. Notably, ASP Chef now supports integration with several high-quality frontend libraries, among them *@vis.js/Network* for interactive network diagrams. These capabilities allow users to construct rich, interactive visualizations tailored to the structure of ASP outputs, making the tool especially useful for teaching, demonstration, and domain-specific analytics. The use of templating decouples the visualization logic from ASP reasoning, promoting reusability and composability of recipes. Our work extends ASP Chef with the capability to ingest DTDL models, enabling a novel interplay between standard digital twin representations and rule-based reasoning.

In summary, while prior works have addressed model-based design, digital twin representation, or ASP reasoning in isolation, this paper contributes a novel integration that combines semantic modeling (via DTDL), logical reasoning (via ASP), and interactive visualization (via ASP Chef), offering a complete pipeline for digital twin analysis and validation.

3. Goal of the research

The goal of this research is to extend the integration of Digital Twins and Answer Set Programming by enabling full lifecycle reasoning and advanced analytics within ASP Chef. Building upon the current support for DTDL model parsing and visualization, we envision the following developments:

- **Automated Instance Management:** support for the creation, validation, and retirement of digital twin instances based on declarative rules and contextual information (e.g., usage patterns, spatial or temporal conditions).
- **Temporal and Predictive Reasoning:** incorporation of temporal extensions to ASP to allow reasoning over telemetry streams, identifying system dynamics, and enabling simulation or forecast of future behaviors.
- **Scalability and Distribution:** adoption of distributed ASP strategies and modular representations to handle large-scale digital twin networks, particularly in domains such as smart agriculture, energy systems, and industrial IoT.

The overarching objective is to create a unified, logic-based framework that supports both structural and behavioral aspects of digital twin systems, offering explainability, constraint checking, and decision support.

4. Current status of the research

This research presents a preliminary result on our ongoing effort to extend ASP Chef with support for the Digital Twins Definition Language (DTDL). Our goal is to enable reasoning and visualization capabilities over digital twin models by integrating a widely adopted semantic modeling language with the declarative power of Answer Set Programming. To this end, we defined a mapping from DTDL constructs (such as interfaces, properties, telemetry, commands, and relationships) to ASP facts, capturing the structure and semantics of digital twin models in a logic-programmable format. This mapping has been implemented as a new operation within ASP Chef, which processes Base64-encoded DTDL models and emits a corresponding set of ASP facts. The new functionality integrates seamlessly with other ASP Chef operations. It supports model querying, constraint validation, and the generation of interactive visualizations through templated configurations of front-end libraries such

as *@vis.js/Network*. The approach is composable and declarative, aligning well with the recipe-based workflow typical of ASP Chef.

A current limitation of our work lies in the absence of support for the serialization of digital twin instances, that is, concrete entities conforming to DTDL models, along with their runtime data. Our goal is to enable full digital twin lifecycle support, from model ingestion to instance management and data-driven reasoning.

5. Preliminary results accomplished

This section illustrates a use case involving the modeling of a vineyard as a system of interconnected digital twins. DTDL is used to define the digital counterparts of key physical components involved in viticulture. The goal is to demonstrate how a complex system composed of sensors, controllers, and monitoring devices can be translated into ASP facts for reasoning and visualization purposes.

5.1. DTDL Interfaces for the Vineyard System

The system is composed of five main DTDL interfaces:

- **Vineyard Property:** represents the entire vineyard estate and acts as the central entry point of the model.
- **Pest Trap:** defines smart traps used for pest monitoring.
- **Soil Moisture Sensor:** describes environmental sensors installed in vineyard plots.
- **Irrigation Controller:** models smart irrigation systems used to manage water distribution.
- **Grape Monitor:** represents devices used to monitor grape development and maturity.

These interfaces define properties, telemetry data, commands, and relationships. For example, the **Vineyard Property** model includes descriptive fields such as name, owner, totalArea, and spatial data (location), as well as relationships to vineyard plots, weather stations, and wineries.

5.2. From DTDL to ASP: A Vineyard Interface

To enable reasoning over digital twin data in ASP Chef, each DTDL model is mapped to a collection of ASP facts. This translation makes it possible to analyze the model declaratively, verifying constraints and inferring system-level properties.

5.3. Modeling System Relationships and Constraints

A key part of the vineyard system lies in the relationships among components. The DTDL models define cardinality constraints and logical links between devices:

- **Vineyard Property** may contain up to 100 **Vineyard Plot** instances.
- **Irrigation Controller** can manage up to 10 plots.
- **Soil Moisture Sensor** and **Pest Trap** devices are installed within individual plots.
- **Grape Monitor** monitors a single plot and provides detailed telemetry.

These relationships are mapped into ASP facts using constructs like:

```
has_relationship("dtmi:...:VineyardProperty;1", "hasVineyardPlots",
                 "dtmi:...:VineyardPlot;1").
maxMultiplicity(("dtmi:...:VineyardProperty;1","hasVineyardPlots"), 100).
target(("dtmi:...:VineyardProperty;1","hasVineyardPlots"),
        "dtmi:...:VineyardPlot;1").
```

```
{ data: {
    nodes: [
      {{= {{f"{ id: '${I}', label: '${N}', group: '${T}' }"}} : node(T,I,N) }} ],
    edges: [
      {{= {{f"{ from: '${F}', to: '${T}', label: '${L}', arrows: 'to' }"}}
        : link(F,T,L) }} ] },
  options: {
    nodes: { shape: "dot", size:20, font:{size:16}, borderWidth:2, shadow: true },
    edges: { width: 2, shadow: true },
    groups: {
      interface: { font: { size: 24 }, size: 30,
        color: { background: lightgreen, border: green } },
      property: { color: { background: yellow, border: orange } },
      telemetry: { color: { background: pink, border: red } },
      object: { shape: "hexagon", color: {background:"#d1c4e9",border: "#512da8"}},
      enum: { shape: "diamond", color: { background:"#fff9c4", border:"#fbc02d" }}
    } } }
```

Figure 1: ASP Chef Mustache template (snippets) to configure *@vis.js/Network* for visualizing DTDL models

This structure supports the application of inference rules for validating instance data and ensuring conformance with the intended digital twin architecture.

5.4. Reasoning and Simulation

The resulting ASP knowledge base allows for reasoning tasks such as:

- Verifying that each **Vineyard Plot** is assigned a moisture sensor.
- Ensuring that no **Irrigation Controller** exceeds its control limit.
- Identifying unmonitored plots or plots with conflicting sensor assignments.
- Simulating scheduling decisions based on sensor telemetry (e.g., when to irrigate or harvest).

For example, relationships specifying as target an interface model not provided in input can be easily identified with the following rule:

```
node(undefined_interface,T,T) :- target(_,T), not interface(T).
```

The obtained instance of `node/3` can in turn be coupled with a Mustache template extending Figure 1 with the group

```
undefined_interface: { font: { size: 24 }, size: 30,
   color: { background: red, border: darkred } },
```

to highlight in red any undefined interface, as shown in Figure 2. A complete recipe is available at https://asp-chef.alviano.net/s/CILC2025/vineyard.

6. Open issues and expected achievements

To realize our vision, several open issues must be addressed. At present, the framework supports only DTDL models, lacking mechanisms to process and reason over actual instances of digital twins that embody live operational data. Our first objective is to extend the pipeline to include the ingestion of JSON-formatted instances enriched with telemetry values and command states, translating them into ASP facts suitable for reasoning. This will require an extension of the current parsing component and

the definition of an enriched vocabulary of facts, maintaining compatibility with the modular logic of ASP Chef.

An essential requirement is the semantic validation of instance data with respect to the DTDL schema from which it originates. This involves enforcing constraints on type conformance, cardinality, access permissions, and hierarchical relations. To this end, we take inspiration from the SHACL framework[15, 16], which provides a powerful means of validating RDF graphs against structural patterns. Our aim is to encode analogous validation logic declaratively in ASP, enabling formal and extensible verification rules grounded in the schema semantics.

Another significant limitation is the absence of temporal reasoning. Digital twin instances often involve time-stamped data and state evolution over time, aspects that cannot be adequately captured through purely static representations. We plan to adopt and adapt existing proposals for temporal extensions of ASP[9, 11], which offer mechanisms for expressing fluents, actions, and temporal constraints. Such extensions are vital for tasks like history-aware validation, detection of behavioral anomalies, and simulation of future system trajectories.

We also intend to integrate predictive components based on statistical learning. Hybrid reasoning—combining symbolic logic with sub-symbolic models—has been shown to be effective in domains requiring both explainability and generalization. In particular, we plan to evaluate the use of time series forecasting tools such as TimeGPT[11] for complementing ASP-based pipelines with trend prediction capabilities. The modularity are key challenge as digital twin ecosystems become increasingly complex. The current framework must be extended with support for distributed reasoning, possibly using federated knowledge bases and modular program decomposition. Caching frequently accessed intermediate results and designing reusable components will be critical for maintaining performance and usability at scale.

Collectively, addressing these challenges will allow ASP Chef to evolve into a comprehensive and versatile platform for digital twin modeling, simulation, and validation, bridging semantic modeling, logical inference, and data-driven decision making in a unified pipeline.

Declaration on Generative AI

During the preparation of this work, the author used ChatGPT to grammar and spelling check. After using this tool, the author reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] M. Alviano, D. Cirimele, L. A. Rodriguez Reiners, Introducing ASP recipes and ASP Chef, in: ICLP Workshops, volume 3437 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023.

[2] M. Alviano, L. A. Rodriguez Reiners, ASP chef: Draw and expand, in: P. Marquis, M. Ortiz, M. Pagnucco (Eds.), Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024, 2024. URL: https://doi.org/10.24963/kr.2024/68. doi:10.24963/KR.2024/68.

[3] C. Dodaro, P. Gasteiger, N. Leone, B. Musitsch, F. Ricca, K. Schekotihin, Combining answer set programming and domain heuristics for solving hard industrial problems (application paper), Theory Pract. Log. Program. 16 (2016) 653–669. URL: https://doi.org/10.1017/S1471068416000284. doi:10.1017/S1471068416000284.

[4] E. Gençay, P. Schüller, E. Erdem, Applications of non-monotonic reasoning to automotive product configuration using answer set programming, J. Intell. Manuf. 30 (2019) 1407–1422. URL: https://doi.org/10.1007/s10845-017-1333-3. doi:10.1007/S10845-017-1333-3.

[5] V. Myllärniemi, J. Tiihonen, M. Raatikainen, A. Felfernig, Using answer set programming for feature model representation and configuration, in: A. Felfernig, C. Forza, A. Haag (Eds.), Proceedings of the 16th International Configuration Workshop, Novi Sad, Serbia, September 25-26, 2014, volume

1220 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2014, pp. 1–8. URL: https://ceur-ws.org/Vol-1220/01_confws2014_submission_14.pdf.

[6] F. Wotawa, D. Kaufmann, Model-based reasoning using answer set programming, Appl. Intell. 52 (2022) 16993–17011. URL: https://doi.org/10.1007/s10489-022-03272-2. doi:10.1007/s10489-022-03272-2.

[7] F. Wotawa, On the use of answer set programming for model-based diagnosis, in: H. Fujita, P. Fournier-Viger, M. Ali, J. Sasaki (Eds.), Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices - 33rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2020, Kitakyushu, Japan, September 22-25, 2020, Proceedings, volume 12144 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 518–529. URL: https://doi.org/10.1007/978-3-030-55789-8_45. doi:10.1007/978-3-030-55789-8_45.

[8] A. A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, E. C. Teppan, Industrial applications of answer set programming, Künstliche Intell. 32 (2018) 165–176. URL: https://doi.org/10.1007/s13218-018-0548-6. doi:10.1007/S13218-018-0548-6.

[9] V. Lifschitz, Answer set programming and plan generation, Artif. Intell. 138 (2002) 39–54. URL: https://doi.org/10.1016/S0004-3702(02)00186-8. doi:10.1016/S0004-3702(02)00186-8.

[10] V. Nguyen, V. L. Stylianos, T. C. Son, W. Yeoh, Explainable planning using answer set programming, in: D. Calvanese, E. Erdem, M. Thielscher (Eds.), Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020, 2020, pp. 662–666. URL: https://doi.org/10.24963/kr.2020/66. doi:10.24963/KR.2020/66.

[11] T. C. Son, E. Pontelli, M. Balduccini, T. Schaub, Answer set planning: A survey, CoRR abs/2202.05793 (2022). URL: https://arxiv.org/abs/2202.05793. arXiv:2202.05793.

[12] O. Cliffe, M. D. Vos, M. Brain, J. A. Padget, ASPVIZ: declarative visualisation and animation using answer set programming, in: ICLP, volume 5366 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 724–728.

[13] R. Lapauw, I. Dasseville, M. Denecker, Visualising interactive inferences with IDPD3, CoRR abs/1511.00928 (2015).

[14] C. Kloimüllner, J. Oetsch, J. Pührer, H. Tompits, Kara: A system for visualising and visual editing of interpretations for answer-set programs, in: INAP/WLP, volume 7773 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 325–344.

[15] P. Hayes, B. McBride, RDF Semantics, W3C Recommendation, World Wide Web Consortium (W3C), 2004. URL: https://www.w3.org/TR/rdf-mt/.

[16] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation, World Wide Web Consortium (W3C), 2014. URL: https://www.w3.org/TR/rdf11-datasets/.

[17] M. Lanthaler, C. Gütl, On using json-ld to create evolvable restful services, International Workshop on RESTful Design (2012) 25–32. doi:10.1145/2307819.2307827.

[18] Azure, Digital twins definition language (dtdl) - version 4 specification, 2024. URL: https://github.com/Azure/opendigitaltwins-dtdl/blob/master/DTDL/v4/DTDL.Specification.v4.md, last updated: 2024-01-20.

[19] Digital Twin Consortium, Dtdl parser, GitHub repository, 2023. URL: https://github.com/digitaltwinconsortium/DTDLParser, open source software for parsing the Digital Twin Definition Language.

[20] Azure, Open digital twins definition language (dtdl) ontologies for buildings, 2020. URL: https://github.com/Azure/opendigitaltwins-building, last updated: 2023-11-15.

[21] Azure, Open digital twins definition language (dtdl) ontologies for energy grids, 2020. URL: https://github.com/Azure/opendigitaltwins-energygrid, last updated: 2023-09-20.

[22] A. V. Aho, M. S. Lam, R. Sethi, J. D. Ullman, Compilers: Principles, Techniques, and Tools, 2 ed., Addison-Wesley, Boston, MA, USA, 2006.

[23] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, P.-A. Champin, N. Lindström, JSON-LD 1.1: A JSON-based Serialization for Linked Data, W3C Recommendation, World Wide Web Consortium (W3C), 2020. URL: https://www.w3.org/TR/json-ld11/#relationship-to-rdf.

[24] Microsoft, Digital twins definition language (dtdl) - version 4, 2024. URL: https://azure.github.io/opendigitaltwins-dtdl/DTDL/v4/DTDL.v4.html, accessed on March 30, 2025.
[25] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, Asp-core-2 input language format, Theory and Practice of Logic Programming 20 (2019) 294–309. URL: http://arxiv.org/abs/1911.04326http://dx.doi.org/10.1017/S1471068419000450. doi:10.1017/S1471068419000450.
[26] M. Gelfond, V. Lifschitz, Logic programs with classical negation, in: D. Warren, P. Szeredi (Eds.), Logic Programming: Proc. of the Seventh International Conference, 1990, pp. 579–597.
[27] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, ASP-Core-2 input language format, Theory Pract. Log. Program. 20 (2020) 294–309. URL: https://doi.org/10.1017/S1471068419000450. doi:10.1017/S1471068419000450.
[28] M. Alviano, W. Faber, L. A. Rodriguez Reiners, ASP Chef grows Mustache to look better, 2025. URL: https://arxiv.org/abs/2505.24537. arXiv:2505.24537.

A. Complex schema visualization example

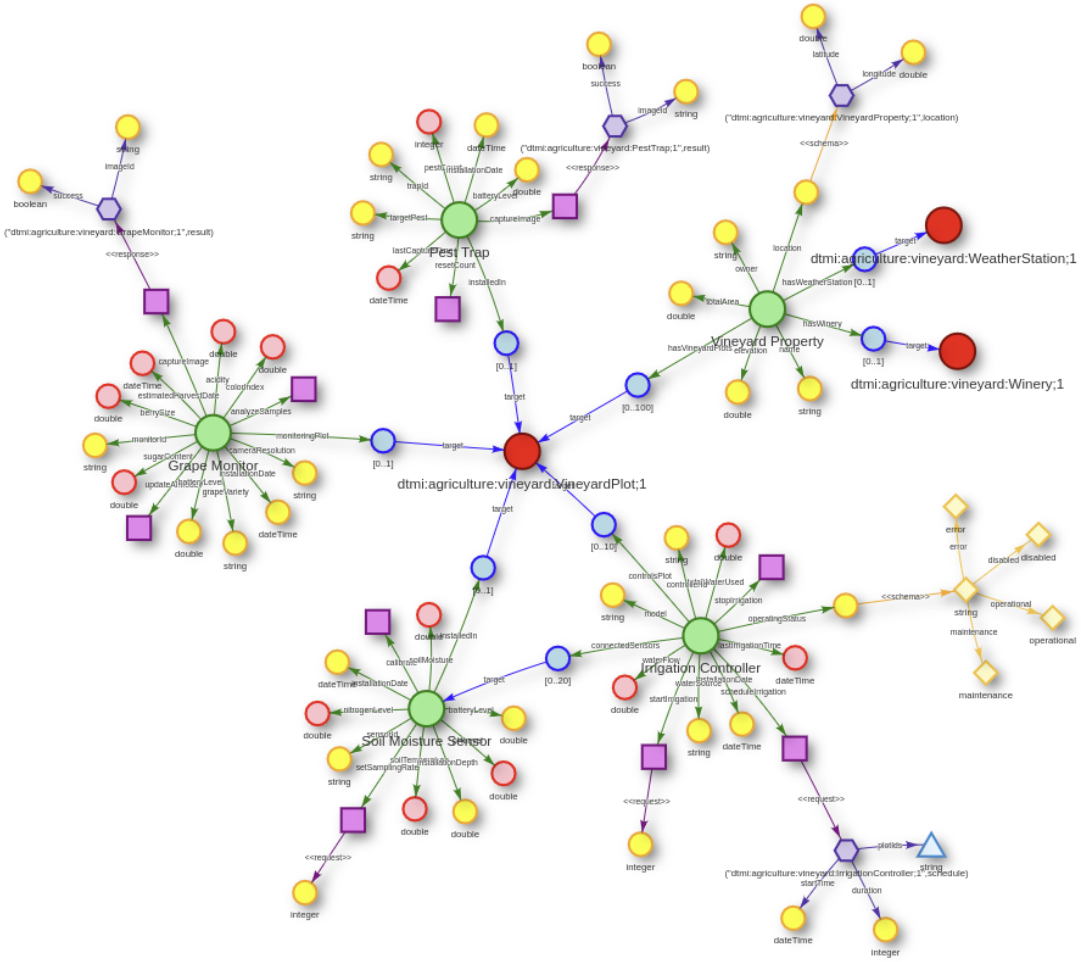

Figure 2: DTDL models of the vineyard use case, with undefined interfaces highlighted in red

In the figure 2 we see a complex pattern of digital twins representing a vineyard. The main elements of the graph are the nodes representing the digital twins of:

- Grape Monitor
- Soil Mosture Sensor
- Irrigation Controller
- Pest Trap

From each node come arcs representing the properties of the individual entities and the type of data used. Finally, it is possible to visually verify which of these expected properties, defined in the models, are not actually defined in the specific instances of the digital twin we are visualising. Visualisation also makes it possible to inspect complex properties and types: properties that refer to an object and not to a primitive type.