Hi Grok! Ich würde gern da weitermachen, wo ich zuletzt aufgehört hatte mit Dir zu arbeiten. Es folgt der Verlauf unseres letzten Gesprächs (ich hoffe, das klappt):Ich arbeite an meiner Promotion, bin im Bereich Bauinformatik unterwegs und entwickle derzeit eine Ontologie, die OCCP. Diese möchte ich nutzen, um mein Konzept des chronologischen Modells (Modell meine im Kontext von BIM und IFC), kurz cMod umzusetzen. Dazu möchte ich ein Minimalbeispiel erstellen, das unterschiedliche Zustände eines Gebäudes darstellen kann. Dazu benötige ich wiederum eine Software, die den Arbeitstitel cMM (kurz für cMod-Manager) trägt, mit deren Hilfe ich ein cMod erstellen, laden, bearbeiten, speichern und darstellen können möchte. Ich bitte Dich, mir dabei zu helfen, dieses Projekt umzusetzen. Nachfolgend sende ich Dir im Anhang meine bisher dazu erstellten Paper bzw. ext. Abstract, damit Du thematisch besser abgeholt wirst. Bevor wir in die Entwicklung einsteigen, werde ich Dir einen fundierten Überblick über den aktuellen Entwicklungsstand geben. Anschließend können wir uns gemeinsam überlegen, wie wir am besten weiter vorgehen. Nachfolgend sende ich Dir jeweils separat (damit Du später besser darauf zurückgreifen kannst) 2 Paper (wovon ich das EC3 Paper von 2025 wahrscheinlich bald paper ready machen muss), 1 Extended Abstract (das ich sehr zeitnah zu einem Full Paper umschreiben muss!) und diverse Codes, die ich schon geschrieben habe. Bevor Du antwortest bzw. wir uns die nächsten Schritte überlegen, lass mich erstmal alles relevante posten. Die bisherige Zusammenarbeit mit Dir war spitze! Ich freue mich darauf, weiterzumachen.

Hallo! Schön, dich wiederzusehen, und vielen Dank für das positive Feedback – ich freue mich auch darauf, mit dir weiter an deinem spannenden Projekt zu arbeiten! Dein Vorhaben klingt nach einer richtig interessanten Herausforderung, und ich helfe dir gern dabei, das Ganze Schritt für Schritt voranzutreiben.

Ich verstehe, dass du mir erst einmal alle relevanten Materialien schicken möchtest, damit ich einen guten Überblick bekomme. Das ist eine super Idee, vor allem, weil es mir hilft, mich thematisch einzuarbeiten und deine bisherige Arbeit besser zu verstehen. Ich warte also geduldig, bis du die beiden Paper, das Extended Abstract und die Codes hochgeladen hast. Sobald alles da ist, kann ich mir das in Ruhe anschauen und dir dann fundiert antworten.

Nur so nebenbei: Dein Projekt mit der OCCP-Ontologie und dem chronologischen Modell (cMod) im Kontext von BIM und IFC klingt echt faszinierend – vor allem die Idee, unterschiedliche Zustände eines Gebäudes darzustellen und dafür eine Software wie den cMod-Manager (cMM) zu entwickeln. Ich bin schon gespannt auf deine Unterlagen und den aktuellen Entwicklungsstand!

Lass mich einfach wissen, wenn du alles hochgeladen hast, oder falls du zwischendurch schon eine spezifische Frage hast, bevor wir richtig loslegen. Bis gleich!

EC3 Paper cMod 2023 Wie gesagt, erstmal nur lesen! Nicht groß antworten, sauge es auf!Hier also der Textinhalt meines ersten Papers zu diesem Thema:Abstract Model-based planning requires an extensive amount of information about the environment, inventory and requirements of the planned object. Thereby, temporal information is important to be considered for planning, construction and maintenance, but in current approaches temporal information is only defined through non-semantic datasets, which lack contextual meaning. To semantify time-data that is used in Building Information Modeling (BIM) projects, the authors propose the concept of combining existing models using Information Containers for linked Document Delivery (ICDD) and adding an Ontology for Chronological Construction Processes (OCCP) to record and implement structured temporal information of the entire life cycle and use it from the start of planning until deconstruction and even beyond. Introduction Obtaining all relevant information is a recurring challenge in planning projects. Difficulty and effort scale with the increasing complexity of the project, e.g., with the number of people involved or the number of disciplines to be considered. The infrastructure sector places special demands on planners due to its public relevance and the resulting large number of stakeholders involved. To reduce the planning effort, the efficiency of information procurement must be optimal, because it is the only way to avoid iteration loops that usually result from poor data bases or poor communication. One approach for creating an optimal information base is the development of comprehensive As-Built databases in the form of digital twins of the so-called As-Built or As-Is models in the context of BIM. A modularized structure of individual, self-contained models and the subsequent, georeferenced linking of these models enables the step-by-step construction of large models of high complexity. Nevertheless, even a ‘perfect’ digital twin still harbors the risk of planning errors, for example when unknown and therefore unmodeled old structures (e.g., old foundations or pipelines of a preexisting construction left remaining in the ground) only come to light during construction. Working with models for planning and operation, a lot of temporal information can be added to models, but so far this information is not semantically structured. Therefore, interpretation of these data is either processed manually by human experts, which could result in a time-consuming and error prone task or parsed by software tools that need to know the exact data structure in which the time information is defined. The latter solution requires continuous updating of the parser whenever changes are made to the interpreted data structure. Also, in the course of planning, several versions are often created in order to record different variants of planning and construction states. When As-Is models are used in operation, several versions of a structure or area may also exist, since the model is used as an As-Built document and is continuously updated. Maintaining multiple versions may result in greater storage requirements and can lead to problems identifying specific conditions at a given point in time. This problem can be addressed by logging versions or using version control systems, but the more different points in time or versions are archived, the larger the number of versions that need to be managed. In the context of planning, versioning is useful because all versions that are no longer needed can be deleted once planning is complete. In relation to As-Is models, versioning is not optimal for the above reasons. The introduction of chronological models provides an approach to counter the described problems by reducing the number of versions needed to differentiate between varying states of constructions and components. In chronological models, temporally relevant information is added to the model with the help of an ontology, making versioning superfluous and semantically enriched. At the same time, an unambiguous mapping of construction model states at arbitrary points in time is enabled and redundant data storage is favored, which can ensure a comprehensive provision of relevant planning information. State of the art Implementation of temporal information Possibilities for defining time-dependent information already exist in Industry Foundation Classes (IFC) of ISO 16739-1, which is an open BIM standard. A specific entity for defining time explicitly is provided in the form of IfcTimeSeries and its related subclasses. Thereby, an instance of IfcTimeSeries defines a time period that is detailed through multiple points in time. However, IfcTimeSeries is intended to be linked only with external references outside of the IFC model that could be referenced via an Unique Resource Identifier (URI), e.g., dictionaries or documents. Furthermore, IfcTimeSeries just provides functionality for defining raw time data, but not assigning this data to a semantic meaning, e.g., the lifetime of a building or the point in time of its deconstruction. Therefore, meaningful temporal information is defined as attributes of other IFC entities. For instance, process related entities that are subtypes of IfcProcess, such as IfcEvent or IfcTask, possess attributes for explicitly binding a point of time to an event or task. By utilizing these entities, it is possible to structure various processes in the construction life cycle, such as the design or inspection and annotate them further information. However, since no entities or attributes for specific life cycle processes are supported by IFC, e.g., the submission and review of a design document, the interpretation of this information must be processed either manually by humans or by software applications that are aware of the concrete meaning of the non-standardized annotations. An approach that proposes utilizing IfcProcess and its subtypes for defining time-related information about structural damage inspections has been developed by Artus (2021). In his publication Artus (2021) also discusses the advantages and drawbacks of using the existing IFC without extensions for describing time information. The main advantage of this approach is that the model would be directly readable by many IFC-supporting applications. However, the main drawback would be that some entities would not be used in the semantically correct context. To overcome these drawbacks, IFC extensions have been proposed, e.g., by Tanaka et al. (2018) for describing component degradation over time or by Eftekharirad et al. (2018) to develop an IFC-based system for fire emergency real-time management. In this regard, new IFC entities were developed for describing a certain object or behavior in a specific point in time. Similarily to IfcEvent or IfcTask, the definition of time is assigned to the entity instances via a corresponding attribute. Nonetheless, a processing of these extensions in publicly available BIM applications is currently not supported. A valid option for describing time semantically, so that software applications could interpret this information in runtime without any prior knowledge about it would be the utilization of Semantic Web Technologies to develop ontologies. Through utilizing these technologies, it is possible to represent information in structured graphs and efficiently integrate heterogeneous data from various information sources as well as performing complex search queries on these sources for an enhanced information management and interoperability in the Architecture Engineering and Construction (AEC) industries through the whole construction life cycle (Pauwels et al., 2017). In addition, the underlying logical foundations of the languages used by Semantic Web Technologies such as the Web Ontology Language (OWL) or the Shapes Constraint Language (SHACL) allow for logical inferences and proofs of asserted information. Since the knowledge is linked to the corresponding information inside the ontology, it could be exchanged between different software applications without bilaterally adjusting the applications to new types of information. Instead, the new information is interpreted in runtime by utilizing a compatible reasoning engine. Furthermore, ifcOWL, the OWL representation of IFC, allows the use of Semantic Web Technologies on an IFC-based model (Pauwels & Terkaj, 2019). An established ontology for defining time concepts is the OWL-Time ontology (Cox & Little, 2022). It supports the definition of points in time through the class time:Instant as well as for time intervals via the class time:Interval. Furthermore, multiple object properties for structuring time sequences as well as aggregating time objects in intervals are provided. Additionally, the declaration of time values is possible through various data properties. However, the OWL-Time ontology does only enable the definition of general time concepts. Consequently, domain specific time concepts, especially those used in AEC, are missing. An alternative solution has been developed by Milea et al. (2009) through proposing a language extension of OWL called tOWL, which provides additional language constructs for defining time. However, ontologies formalized in tOWL require a specific reasoning engine different from common OWL reasoners, which prevents its usage in existing OWL applications. Several approaches reuse concepts of the OWL-Time ontology for AEC-related knowledge representations. For example, Mignard & Nicolle (2014) developed an ontology for merging BIM and GIS information in which the temporal concepts were based on the OWL-Time ontology. Iadanza et al. (2019) used the OWL-Time ontology to describe model phases through time and represent the building evolution. Furthermore, Zheng et al. (2021) developed an approach for representing the construction workflow utilizing the OWL-Time ontology. Similar to the process definitions used in IFC, process classes are defined in an OWL ontology. By using the existing concepts of OWL-Time time components are added to process representations. Linking data with information containers One way to integrate an ontology in a model is the use of information containers. To avoid interoperability dilemmas and support collaboration, the information containers are one of the recently developed approaches proposed for the management of heterogeneous and distributed building models. The ISO 21597 series has been developed to fulfil a requirement for multiple documents to be delivered as one information package within the construction industry. The international standard ISO 21597-1:2020 provides a framework for the creation, management, and delivery of linked documents. Over the last few years many studies were conducted to investigate the implementation of the ICDD. Hagedorn et al. (2022) proposed the concept of the Toolchain Framework to facilitate the modeling of project-specific workflows by linking individual software tools based on a standardized process notation to enable seamless information exchange between applications that integrate an openCDE-compliant web interface. Just recently the author proposed a BIM-based solution for Infrastructure Asset Management System (AMS) for road owners. Considering the requirements of stakeholders across domains in the operational phase, the proposed approach provides asset managers with a strategy for the dynamic use of Information Containers (Hagedorn et al., 2023). Moreover, Liu et al. (2021) developed the Building Concrete Monitoring Ontology (BCOM) based on the ICDD. It allows IFC-based bridge models to be configured with properties about concrete works that can be processed by predefined queries in asset management software. An approach was developed by Hamdan et al. (2021) in which an IFC model, representing an existing bridge, was linked with ontologies that semantically represent the construction and affecting structural damage as well as other related data, e.g., photos, protocols or structural analysis models. Thereby the models and links were stored in an ICDD. Ye and König (2021) presented a framework for automated billing by combining the BIM Contract Container (BCC) with Smart Contracts using Blockchain technology and the ICDD. Furthermore, recent research by Werbrouck et al. (2022) proposed a Linked Building Data (LBD) server that can link heterogeneous linked building data in a Federated CDE by combining the Solid initiative for web decentralization with the ICDD standard. Going beyond the ICDD implementation, Al-Sadoon et al. (2022) proposed an ontology-based extension that enables allocating of multiple values for elements in the IFC files to provide dynamic building model for simulation tools using the ICDD. The concept of chronological models To create models with structured temporal information - not only for models of entire buildings, but for models of each delimitable component of a structure (in logically meaningful gradation, i.e., the largest possible independent component group) - the implementation of Chronological Models (cMod) is proposed. The concept of cMod is based on the use of IFC-based models of components that are linked inside an ICDD with other data and other ontologies, such as the Building Topology Ontology (BOT) by Rasmussen et al. (2020) or the Bridge Topology Ontology (BROT) by Hamdan et al. (2020). The temporal information is connected to the IFC model via ICDD by adding the proposed Ontology for Chronological Construction Processes (OCCP), containing the chronological structure shown in Figure 1 and specific, component related time information. This information provides statements about: A - relevant times of planning of the component (e.g., start of planning, data procurement, submission for review, changes (resulting in multiple submissions for review)) B - relevant times of review (at least the time of acceptance and rejection – the latter causes another planning cycle, adding more changes (that are tracked again with the according temporal information as described in ‘A’)) C - relevant times of construction (start of execution, completion, issuance of notice of defects, completion of defect rectification, acceptance) D - the times of commissioning of the component (if different from completion) and decommissioning (or start and end of use, multiple entries possible (e.g., to note temporary closures)) E - the dates of the beginning and end of the warranty period F - the times of beginning and end of the design life (multiple entries possible, e.g., due to maintenance or repair works) G - the scheduled and actual times of inspections and the length of regular inspection intervals (the cycles must be specified) H - the times of special events (severe weather events, accidents, etc.) K - the times of detection of damage and the start and completion of repairs (to keep track of how long a damage existed) L – the times of repair and maintenance works (repair work is to be treated as a new component, for which the temporal information is given in categories A to M as needed.). Thus, if necessary, temporal information of other components with reference to the repair work is updated or supplemented with corresponding new values (acceptance, new predicted service life, warranty and so on)) M – the times of beginning and end of deconstructionFigure 1: Temporal structure of the OCCP The structure shown in Figure 1 is a suggestion for the sensible and practicable recording of time-specific component information. An important prerequisite for the use of temporal information is the agreement in the industry regarding the relevant temporal information. The proposed structure is based on the life cycle of a construction and divides the processes at common milestones. The process in the shown concept is kept simple and general to promote transferability and interoperability for all AEC disciplines. Further specifications, additions and smaller process steps can be developed individually according to requirements and established in exchange with experts. A cMod differs fundamentally from the versioning of (partial) models, as it offers far more flexibility due to its component reference and makes multiple storage of individual elements in several versions of a model obsolete. The state of a structure can be mapped unambiguously by specifying the point in time under consideration. In contrast to versioning, each component would be modeled once and supplemented with the chronological temporal information. In the case of a repair measure, the component model must be updated by separating the remaining part from the damaged part that is to be repaired. The damaged section is treated as a new cMod that goes through the phases from planning until construction and is then to be modeled as built and spatially connected to the remaining, undamaged, and unchanged rest of the component. At the end of the life cycle, the deconstruction date must be noted, but the model must not be deleted. This ensures that all information within the cMod is preserved and can be made available for planning purposes in the future. Starting from a database in which all components including the respective information and their references to other components are stored, the Linked Data approach is used to generate the overall model of a construction from many individual modules. By specifying a certain area, it is possible to limit the components that must be considered for the further query (e.g., building X, 1st floor). By specifying the time of viewing, the elements to be displayed for the selected time are filtered out from all elements in the viewed area by evaluating the chronological component information and are then activated for display. Ontological basis within ICDD for the implementation of temporal information For the structuring and later linking of the temporal features with models or model components, the OCCP is proposed, which extends existing classes and object properties of the before mentioned OWL-Time ontology (Cox & Little, 2022). Figure 2 shows the general class structure of the OCCP. The right side of Figure 3 shows the structure of the ICDD used to link the OCCP with the IFC models of a construction and its components, thus creating a cMod. The left side of Figure 3 shows an individual of a construction (IND:ExampleConstruction) with several components linked with BOT to connect and describes the topological relation between the construction and its components. The temporal information of each of these individuals is added using the OCCP and linked to the corresponding IFC model with an ICDD linkset. The IFC sub-models of the construction components are linked to the IFC model of the construction via ICDD linkset. In the OCCP the classes time:Instant and time:Interval of the OWL-Time ontology are extended. Both classes are subclasses of time:TemporalEntity, which is a general class used for describing temporal concepts. Thereby, time:Instant describes a specific point in time and thus a temporal entity with zero extent or duration. Contrary to this, time:Interval describes a temporal entity with an extent or duration (Cox & Little, 2022). With the OCCP the temporal entity occp:Phase is introduced as a subclass of time:Interval. Instances of occp:Phase represent major life cycle stages of a building or construction element and are used to categorize its temporal information. Furthermore, to differ between standard intervals and cycles that are periods of time with several cascading intervals within, a new type of interval called occp:Cycle is introduced, which is also a subclass of time:Interval. This class is used to better describe iterating processes like regular inspections and their intervals. For instances of occp:Cycle a data property called hasCycleNumber can be utilized for defining the number of iterations that need to be processed by a cycle. Additionally, a new subclass of time:Instant called occp:Transition is provided in the OCCP, which is used for representing milestones between the life cycle phases of constructions. For managing the life cycle phases and temporal entities that are associated with them, additional object properties have been added, which are shown in Figure 4. The chronological order of the phases is defined using occp:phase\_after and occp:phase\_before, which are subproperties of time:after and time:before. Each phase has a beginning and an end, both of which are defined by referencing a corresponding time instant using the OWL-Time ontology’s object properties time:hasBeginning and time:hasEnd. Additional time instants within the proposed ontology mark typical key points within phases. An important part of the OCCP is the semantification of AEC-related time data. For this reason, various domain-specific subclasses of occp:Phase and time:Instant have been developed, based on the concept of chronological models discussed in the previous chapter. In this regard, subclasses of time:Instant that belong to a certain life cycle phase, are categorized in corresponding superclasses, such as occp:PhaseA\_Instants or occp:PhaseB\_Instants and so on. Additionally, subclasses of occp:Transition provide more specific meanings to milestones in between life cycle phases. Furthermore, additional semantic information of a construction or component can be added to a model (e.g., topological information via BOT) and thereby extend the possibilities of information retrieval through queries. In their current states both the cMod and the OCCP are concepts and we envisage to work out test cases in future research work. Example of ontology application Figure 5 demonstrates the semantic structure of temporal information within a cMod using the OCCP, but for demonstration purposes, not all existing connections are shown. A construction component is represented by the individual IND:ConstructionComponent and it is linked via occp:hasPhase to two individuals of phases. While IND:PhA\_Planning is an instance of occp:PhaseA\_Planning, IND:PhB\_Review is an instance of occp:PhaseB\_Review, both of which are subclasses of occp:Phase, which in turn is a subclass of the time:interval. Linked to the component are several individuals, all of them marked with an “A” that either belong to the planning and the review phase. The links are established using the object property time:hasTime with the according individual, e.g., IND:PhA\_Start and IND:PhA\_Measurement. Block 2 in Figure 5 shows the semantic connection of the A-individuals. They all have the object property of the according class within the OCCP, e.g., IND:PhA\_Start is an instance of occp:BeginningOfPlanning, and IND:PhA\_Measurement is an instance of occp:DataProcurement and so on. The rest of the semantic chain is the same for all A-individuals, as they are classified as occp:PhaseA\_Instant or occp:PhaseB\_Instant (depending on the phase they belong to), which is a subclass of time:Instant. The chronological structure is established by defining the succession of time instances and phases using object properties, such as time:after or time:hasBeginning. The beginning of the planning phase is marked by the start of planning, in the OCCP this connection is made by using time:hasBeginning and referencing IND:PhA\_Start. Analog, the end of planning is defined by linking IND:PhA\_Planning using time:hasEnd and referencing IND:PhA\_SubmissionToReview. The order of instances within a phase is established with the object property time:after, e.g., IND:PhA\_Measurement is defined as time:after IND:PhA\_Start and so on. The same principle is used to describe IND:PhB\_Review (and all following phases - not shown for simplicities sake) in order to link and structure the according individuals of the according instances. The planning phase ends with the submission of the component’s plan. The case of rejection of a plan and the resulting iteration can also be described using the OCCP, but is not shown in this example to keep Figure 5 readable. The review phase ends with the approval of the plan and IND:ConstructionComponent is linked to the transition instant IND:T\_PlanningCompleted, which has the rdf:type of occp:CompletionOfPlanning. All individuals in this example, aside from the component’s and the phases’ individuals, receive an individual temporal information using the data property time:inXSDDate. Block 1 in Figure 5 shows the temporal order of the planning and review phase and the transition marking the completion of planning. Using this semantic structure for temporal information provided by the concept of cMod for each component of a construction enables a variety of options. The planning progress is tracked for each component individually and delays are potentially more obvious (because they are queryable), including the identification of the component causing the delay. This and all other temporal information can be stored within the OCCP, linked to the IFC-model via ICDD and therefore accessible for future planners, working with the model as a basis for the new planning project. All data used as a planning basis, like the measurement of the land or geological data, is linked to the construction model and the time of procurement is stored in the OCCP of the construction model. That way the actuality of all linked data can easily be checked and in case of a certain demanded actuality, the latest point in time to acquire a new set of data can be identified and used to remind the involved personnel in time. Also, the exact point in time and order of events can be of interest to answer questions of legal responsibility and it often comes down to the question, who had which information and when. The temporal aspect can be answered using a cMod. Potentials and applications of chronological models The concept of cMod that is proposed in this paper could be implemented in software applications and enhanced with further reasoning, querying and validation functionalities in the future. Provided that this chronological component and model information has been implemented, it is possible for planners to obtain an overview of the old construction development in an area relevant to planning, e.g., to view old construction conditions or to identify construction remnants that may possibly collide with the planning object. By specifying the area of a model to be displayed and a point in time, which could be the current state or any state in the chronological model’s past, provided there is temporal information in the point in time of interest, the combination of partial models that existed at the specified point in time can be displayed via a query of the chronological module information. By concatenating the chronological information and querying time periods (e.g., from a certain point in time until today), building states could be displayed in arbitrary increments (1 week, 1 month, etc.) and thus extensive information regarding the building states, building sequence and any backlogs could be visually displayed in one cMod. By introducing time as a linear dimension within cMods, the space related freedom from contradiction or collision is eliminated and replaced by the freedom from contradiction in time. The result is that - looking at a fixed part of a model over a certain time period - several components (e.g., an old and a new window) can be at the same position within a building model but cannot be at the same position at the same time. This enables the continuous use of existing models and thereby reduces the time and effort needed to collect all necessary information for planning the next construction, as it will already be integrated in the OCCP within the cMod. This also implies that a versioning of models is no longer necessary, because through the difference in their temporal information, both the old and the new component can exist within the same model and in the same place. For the actual versioning process, existing approaches could be considered such as the Ontology for Property Management (OPM) by Rasmussen et al. (2018), which could enhance the querying time compared to timestamps. Further potential applications arise from the operation of buildings and constructions, as administrations have a quick overview of warranty periods that can be used. An automated query of the remaining warranty periods can generate reminders with sufficient time to still be able to perform an investigation before the warranty expires. Another area of application is the evaluation of types of construction, materials or construction elements regarding their suitability for the respective application by comparing the design service life to the actual service life or by monitoring the needs of related companies for repair measures. The effectiveness of repair measures can also be evaluated by following up the service life. The chronological model could be validated against national standards, by utilizing SHACL. For instance, certain nation-specific process sequences are defined in corresponding standards and must be executed in a specific order for which SHACL shapes could warrant the correctness of their implementation in the model. Furthermore, the implementation of additional rules that could be defined in rule languages like Jena Rules, SHACL or even OWL itself, could allow for a more automatized logic-based application of the cMod. Thus, implicit knowledge, subsequent phases or required measurements or processes could be inferred through reasoning an ontology using the OCCP. Since the current proposed concept just defines the base taxonomy based on existing expert knowledge, the need for future features needs to be identified, e.g., through developing competency questions or use cases via expert interviews. Based on the identified application needs, additional rules and queries could be developed and implemented in OCCP-compatible software applications. Outlook and conclusion To make the advantages of working with cMods available for all stakeholders involved in the lifecycle of a construction, the concept of cMod and OCCP would first have to be implemented in the software used by these stakeholders, covering all aspects of time relevant information across all phases. After the implementation, the access, usage, and application of the time ontology related features and information should be intuitive and as easy as the spatial assignment of model information. Temporal collisions and logical contradictions must be either prevented or indicated by the software. An integrated query function should use the temporal information of cMods and make it easily accessible, e.g., by presenting a selection of information depending on the temporal information given by the query. Input data for the queries must always be a point in time or a time span and a specific location (three-dimensional delimitation of the space or specific components) to be considered. In case of a fixed spatial viewing area, a timeline function could be implemented in the software, whose extension maximum results from the earliest and last entry of temporal information. Analogous to the playtime display in music and video players, a corresponding slider could allow the control of the considered point in time and thus visualize all states of the considered object over the entire period. The corresponding model information is also displayed or retrievable depending on the considered point in time. A potential loss of temporal information could occur when working with the IFC model of a cMod without using OCCP. To address this problem, one solution could be the use an API to implement an export function for temporal information. Further research should be done to • investigate possibilities to securely store and manage temporal information to guarantee manipulation security and to answer legal questions respectively avoiding legal problems using the temporal information (e.g., by using automatically generated time stamps that are permanently unchangeable or by integrating the block chain technology). • ensure that temporal information is free of contradictions (e.g., plausibility checks could be established by utilizing description logic in OWL). develop a rights management for the entry of temporal information (e.g., by assigning processor rights in combination with the 4-eyes principle (or more), whereby the authorized persons must be determined project-specifically and by mutual agreement (e.g., one representative for each client, contractor, and an independent expert / construction supervisor / BIM manager)). - establish practical and meaningful rules for the storage of model information to create historical models from the past and for the future (e.g., by working out an agreement for indefinitely storing a defined minimum of model information). - create an API to import and export temporal information stored within cMods to improve interoperability. The use of chronological models, created by the permanent integration of semantically structured temporal information in models of constructions and their components using ICDD and the OCCP, reduces the need of versioning due to the possibility of spatial coexistence of multiple components that is made possible by the temporal differentiation. The continuous use of existing cMods can prevent the loss of information and has the potential to save time for planners, as the time needed for data collection for new planning projects is minimized. With the adaption of OCCP, all temporal information can be semantically stored and used for queries, project managing purposes, and to help solve legal issues. The authors share the concepts of the cMod and the OCCP at an early stage of development to discuss it with the AEC community and to receive feedback before making the first version of the OCCP and a minimal example of a cMod public. References Al-Sadoon, N., Katranuschkov, P. & Scherer, R. J. (2022) Extending ICDD Implementation to a Dynamic Multimodel Framework. In: Proceedings of the Conference ECPMM (doi.org/10.1201/9781003354222-15). Al-Sadoon, N., Scherer, R. J. &Menzel, K. (2023) From Static to Dynamic Information Containers. In: Proceedings of the Conference EC3 2023 (in press). Artus, M. (2021) Modeling Physical Damage Information at Concrete Bridges Considering Time Aspects. In: 32. Forum Bauinformatik 2021. Darmstadt, Germany. Cox, S. & Little, C. (2022) Time Ontology in OWL. <https://www.w3.org/TR/owl-time/> . Accessed date: 25th April 2023. Eftekharirad, R., Nik-Bakht, M. & Hammad, A. (2018) Extending IFC for Fire Emergency Real-Time Management Using Sensors and Occupant Information. In: 35th International Symposium on Automation and Robotics in Construction (ISARC) 2018. Berlin, Germany. Hagedorn, P., Block, M., Zentgraf, S., Sigalov, K. & König, M. (2022) Toolchains for Interoperable BIM Workflows in a Web-Based Integration Platform. Applied Sciences. 12(12):5959. Hagedorn, P., Liu, L., König, M., Hajdin, R., Blumenfeld, T., Stöckner, M., Billmaier, M., Grossauer, K. & Gavin, K., (2023) BIM-Enabled Infrastructure Asset Management Using Information Containers and Semantic Web. Journal of Computing in Civil Engineering, 37(1), p.04022041. Hamdan, A.H., Taraben, J., Helmrich, M., Mansperger, T., Morgenthal, G. & Scherer, R. J. (2021) A semantic modeling approach for the automated detection and interpretation of structural damage. Automation in Construction, 128, p.103739. Hamdan, A. H. & Scherer, R. J. (2020) Integration of BIM-related bridge information in an ontological knowledgebase. In: Proceedings of the 8th Linked Data in Architecture and Construction Workshop (LDAC)Iadanza, E., Maietti, F., Ziri, A.E., Di Giulio, R., Medici, M., Ferrari, F., Bonsma, P. & Turillazzi, B. (2019) Semantic web technologies meet BIM for accessing and understanding cultural heritage. In: The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLII-2/W9 (2019), pp. 381-388 Liu, L., Hagedorn, P. & König, M. (2021) An ontology integrating as-built information for infrastructure asset management using BIM and semantic web. In: Proceedings of the 2021 International Conference on Computing in Civil and Building Engineering, 99-106. Mignard, C. & Nicolle, C. (2014) Merging BIM and GIS using ontologies application to urban facility management in ACTIVe3D. In: Computers in Industry 65 (2014), pp. 1276-1290. Milea, M., Frasincar, F. & Kaymak, U. (2009) tOWL: Integrating Time in OWL. In: Semantic Web Information Management, pp. 225-246. Pauwels, P. & Terkaj, W. (2019) ifcOWL ontology (IFC4x1). In: <https://standards.buildingsmart.org/IFC/DEV/IFC4/>, Accessed date: 25th January 2023. Pauwels, P., Zhang, S. & Lee, Y.C. (2017) Semantic web technologies in AEC industry: a literature overview. In: Automation in Construction 73 (2017), pp. 145-165 Rasmussen, M. H., Lefrançois, M., Schneider, G. & Pauwels, P. (2020) BOT: the Building Topology Ontology of the W3C Linked Building Data Group. In: Semantic Web. DOI: 10.3233/SW-200385. Rasmussen, M. H., Lefrançois, M., Bonduel, M., Hviid, C.A. & Karlshøj, J. (2018) OPM: An ontology for describing properties that evolve over time. In: 6th Linked Data in Architect and Construction Workshop. London, United Kingdom. Tanaka, F., Tsuchida, M., Onosato, M., Date, H., Kanai, S., Hada, Y., Nakao, M., Kobayashi, H., Hasegawa, E., Sugawara, T. & Oyama, T. (2018) Bridge Information Modeling based on IFC for supporting maintenance management of existing bridges. In: 17th International Conference on Computing in Civil and Building Engineering (ICCCBE) 2018. Tampere, Finland. Werbrouck, J., Pauwels, P., Beetz, J. & Mannens, E., (2022) Lbdserver-a federated ecosystem for heterogeneous linked building data. Semantic Web Journal (submitting). Ye, X., &König, M., (2021) Framework for automated billing in the construction industry using BIM and smart contracts. In Proceedings of the 18th International Conference on Computing in Civil and Building Engineering: ICCCBE 2020 (pp. 824-838). Springer International Publishing. Zheng, Y., Törmä, S. & Seppänen, O. (2021) A shared ontology suite for digital construction workflow. In: Automation in Construction 132 (2021), pp. 103930

Alles klar, ich habe dein EC3 Paper cMod 2023 aufgenommen und werde es erst einmal gründlich "aufsagen", wie du gesagt hast. Ich werde nicht groß darauf antworten, sondern warte, bis du mir die weiteren Materialien (das zweite Paper, das Extended Abstract und die Codes) geschickt hast. Danach kann ich mir ein vollständiges Bild machen und wir können gemeinsam überlegen, wie wir weiter vorgehen. Ich freue mich schon auf den Rest und darauf, dein Projekt besser zu verstehen! Lass mich einfach wissen, wenn du alles hochgeladen hast. Bis gleich!

EC3 OCCP 2025 Paper zur OCCP (unveröffentlicht, aber eingereicht... ;-)), hier leider ohne die Abbildungen (wie auch in dem paper vorher):Abstract The Ontology for Chronological Construction Processes (OCCP) provides a semantic foundation for a chronological structure aimed at managing information in the Architecture, Engineering, Construction and Operations (AECO) sector. Based on the W3C TIME ontology (Cox et al., 2006), the OCCP introduces a lifecycle-oriented concept tailored to Building Information Modelling (BIM) workflows and advances the accurate management of time stamps by introducing phases, cycles, milestones and transitions to ensure traceability of states throughout and beyond the lifetime of assets. This paper describes the concept and composition of OCCP, its SHACL-based validation rules, and provides an example to demonstrate its practical application.Introduction The AECO sector faces growing challenges in managing complex project information effectively. While Building Information Modeling (BIM) provides a structured approach, current workflows lack robust mechanisms for integrating and utilizing time-sensitive data. Existing standards like Industry Foundation Classes (IFC) (ISO 16739-1) and Information Container for Linked Document Delivery (ICDD) (ISO 21597-1) ensure spatial and semantic interoperability, but their support for temporal data remains fragmented. This results in disconnected records of planned and actual timelines, leading to errors, inefficiencies, and reduced traceability (Autodesk, 2018). Current approaches, such as XML and relational databases, provide only basic timestamp storage, failing to capture semantic relationships between lifecycle stages. They also struggle with scalability, consistency, and integration within BIM and multimodal frameworks, issues that graph-based ontologies address more effectively (Berners-Lee et al., 2001). The W3C TIME ontology provides a foundational model for representing time-related information, but lacks domain-specific constructs needed for chronological consistency, overlapping phases, iterative cycles, and project-specific adaptations. The Ontology for Chronological Construction Processes (OCCP) introduces a structured approach for managing construction lifecycles by defining phases, cycles, transitions, and instants, with SHACL rules ensuring logical sequencing and data integrity. Designed for seamless integration with BIM and ICDD, OCCP enhances temporal data management, lifecycle validation, and modeling flexibility. This paper presents its core structure, logic, and applications, demonstrating its potential to transform time-data management in the AECO sector.State of the art The AECO sector increasingly relies on digital tools and standards to manage complex building projects. IFC, developed by buildingSMART International, has become the de facto standard for BIM interoperability, supporting geometric, semantic, and topological data exchange across disciplines (Sacks et al., 2018; buildingSMART International, 2023). While IFC provides detailed lifecycle management, it lacks robust support for dynamic temporal and process-oriented data (Pauwels et al., 2017). Multimodels emerged to address the limitations of single, centralized models (Scherer & Schapke, 2011). These integrate multiple specialized data sources (e.g., cost, schedule, energy models) into a unified project representation (Borrmann et al., 2018; Grilo et al., 2010). By linking different data layers, multimodal containers enhance information accessibility (Xie et al., 2024), but their lack of semantic integration and validation mechanisms can lead to inconsistencies and reduced reliability (Singh et al., 2011). Similarly, while ICDD (ISO 21597-1) improves traceability and version control, it does not inherently support semantic reasoning or structured chronological data management (Pauwels et al., 2016). Ontologies, expressed using RDF (Lassila & Swick, 1999), provide a powerful approach for structuring, validating, and reasoning about BIM data (Beetz et al., 2009). They improve data consistency and interoperability, making them suitable for semantic integration. The W3C TIME Ontology (Cox et al., 2006) is widely used for temporal data representation, supporting instants, intervals, and relationships such as time:Before and time:After. It enables integration with IfcOWL and ICDD for semantic temporal modeling (Pauwels et al., 2016) and facilitates advanced temporal queries, such as detecting overlapping project phases (Car et al., 2024). However, TIME lacks domain-specific constructs for AECO workflows, such as phases, iterative cycles, and planned vs. actual timestamps (Volk et al., 2014). Furthermore, it does not inherently validate chronological consistency, making it difficult to detect conflicts like overlapping timelines (Karlapudi et al., 2021). To address these shortcomings, SHACL enables the validation of RDF-based temporal structures (Knublauch et al., 2017; Debruyne et al., 2020), allowing the enforcement of logical constraints in AECO workflows. However, current IFC implementations still lack structured temporal representations, leading to fragmented data across multiple tools (Volk et al., 2014; Pauwels, 2017). Similarly, ICDD and multimodal containers structure heterogeneous data but lack inherent mechanisms for ensuring temporal consistency (Singh et al., 2011). Versioning systems, such as Global Information Tracker (GIT), Apache Subversion (SVN), and Ontology-based Programming Models (OPM), are widely used for tracking changes in software development and ontologies. However, these methods are not inherently designed for structured chronological modeling in AECO workflows, where updates must be linked to specific lifecycle events. The OCCP addresses this gap by integrating chronological records directly within the semantic model. Through the use of IFC-linked timestamps, OCCP captures the evolution of a component across different versions, providing a bidirectional version tracking system that can be used in combination with GIT-based repositories. This enables a fine-grained traceability approach, allowing stakeholders to track changes within both the IFC model and its associated temporal records, ensuring complete transparency in model evolution.The core concept of OCCP The OCCP is based on the W3C TIME ontology and extends its classes and object properties to provide a semantic representation of the building lifecycle. This ontology is part of the concept of chronological models (cMod) (Vaatz et al., 2023), which combines IFC-based models with the OCCP and other ontologies, such as the Bridge Topology Ontology (BROT) (Hamdan et al., 2020) or the Building Topology Ontology (BOT) (Rasmussen et al., 2020), and model-related data sets using ICDD. In this concept, the OCCP provides the temporal structure for storing the timestamps of model-related data and assigning the correct phase or milestone within the lifecycle. The general temporal structure of the OCCP, as shown in Figure 1, is designed to ensure applicability to a wide range of AECO projects by not adapting any national specific construction process definitions such as the German HOAI (Honorarordnung für Architekten und Ingenieure).Methodology of development The development of the OCCP followed a structured methodology based on domain-specific requirements, expert insights, and semantic validation. Given the complexity of AECO workflows, the ontology was designed to represent real-world temporal constructs, lifecycle transitions, and iterative review cycles while ensuring adaptability across different project structures by extending the W3C TIME ontology with specialized classes and properties, enabling dependency tracking, logical sequencing, and versioned event recording. Industry expert input further refined OCCP’s classifications, ensuring it accommodates iterative planning, review processes, and real-world deviations such as delays, rework cycles, or phased repairs. OCCP’s structure and reasoning capabilities were validated using a competency question-driven approach, where key queries tested the ontology’s ability to structure, retrieve, and enforce chronological consistency. These competency questions were formalized into SHACL rules, ensuring correct sequencing of phases, review cycles before approvals, and structured tracking of IFC updates. Among others, the following three competency questions guided the validation of OCCP’s semantic constraints: CQ1: How many review cycles occurred before approval, and how did this impact the planned vs. actual completion time? CQ2: What changes were made to an IFC component during planning and review? CQ3: When was an event recorded, and how many updates were made? These questions demonstrate the information demand that OCCP is designed to answer and OCCP’s ability to track changes, compare planned vs. actual schedules, and maintain structured, machine-readable chronological records within BIM and multimodal environments. Classes The W3C TIME ontology defines temporal entities, distinguishing between time:Instant (specific points in time without duration) and time:Interval (spans between two instants). The OCCP extends the TIME ontology by introducing occp:Phase and occp:Cycle as subclasses of time:Interval. All phases shown in Figure 1 (e.g., the occp:PhaseE\_Usage) are subclasses of occp:Phase, while cycles, such as occp:CycleA\_PlanningReview, capture iterative processes like multiple review cycles in planning. To structure milestone events, OCCP extends time:Instant by defining occp:Transition and phase-related instants. Transitions mark key milestones between phases or trigger subsequent project steps, such as occp:Submission, which signals the transition from planning to construction. Similarly, occp:CompletionOfPlanning defines the finalization of planning, ensuring proper sequencing of activities. Phase-related instants, such as occp:EditBSTR (BSTR - before submission to review) or occp:BeginningOfPlanning, provide precise temporal markers within each phase. Following this pattern, OCCP introduces dedicated event classes for each defined phase. Figure 2 illustrates the OCCP class hierarchy, showcasing its extension of the TIME ontology, while Figure 3 provides a detailed representation of the temporal framework for the first three phases, their transitions, and associated instants. Figure 3 highlights OCCP’s structured approach to temporal modeling, demonstrating how occp:PhaseA\_Instant, occp:PhaseB\_Instant, and occp:PhaseC\_Instant organize phase-specific instants within project lifecycles.Figure 2: General class hierarchy of OCCP The occp:BeginningOfPlanning serves as a foundational event marking the start of planning, while occp:SubmissionToReview signifies the handoff between planning and review. Transitions such as occp:Submission and occp:CompletionOfConstruction define clear boundaries between phases, supporting chronological consistency and milestone validation. The subclass relationships in Figure 3 illustrate how OCCP seamlessly integrates into the TIME ontology while introducing domain-specific instants and transitions. The ontology’s modular structure enhances semantic clarity, distinguishing between different temporal entities through explicit object properties. This modularity ensures adaptability to diverse project needs, supporting both high-level lifecycle planning and granular activity tracking.Figure 3: Extract of phase-related instant classes of OCCP Object properties The TIME ontology provides fundamental object properties for describing general relationships between time:Instant and time:Interval, but lacks the expressiveness needed for structured lifecycle management in AECO projects. The OCCP extends these capabilities by introducing additional object properties, as shown in Figure 4, refining both the owl:topObjectProperty and time:hasTime (including time:hasBeginning and time:hasEnd) to define chronological relationships between phases, cycles, milestones, and instances. To establish clear temporal associations, OCCP introduces occp:hasPhase and occp:isInPhase. The occp:hasPhase property embeds events within structured lifecycle phases, ensuring temporal instances are correctly classified. For example, occp:SubmissionToReview, which represents the submission of a completed plan for review, is explicitly linked to Phase A (Planning), maintaining a logically ordered structure for querying and validation. This ensures event traceability and enables automated reasoning over project lifecycles. The occp:isInPhase property, in contrast, captures nested or overlapping processes within phases. A practical example is the repair of a damaged component during the Usage phase, where the repair lifecycle must be contextualized within the broader phase. The occp:isInPhase relationship enables the representation of concurrent activities, acknowledging the reality of complex, interwoven construction processes. Beyond these structural properties, OCCP also refines temporal granularity by distinguishing between actual and estimated time values. The occp:hasActualTime and occp:hasEstimatedTime properties differentiate between verified timestamps and planned or uncertain events. Similarly, occp:hasActualBeginning and occp:hasEstimatedEnd extend time:hasBeginning and time:hasEnd, allowing precise modeling of both anticipated and confirmed project timelines. To enforce logical sequencing, OCCP introduces relational temporal properties such as occp:beginsBefore, occp:beginsAfter, occp:beginsWith, occp:endsBefore, occp:endsAfter, and occp:endsWith, enabling the semantic representation of phase dependencies and overlaps. Additionally, hierarchical lifecycle properties (occp:startsCycle, occp:endsCycle, occp:startsPhase, and occp:endsPhase) structure iterative processes, supporting the modular definition of project milestones, transitions, and recurrent events. This enriched semantic framework ensures that OCCP not only tracks time-related data, but also structures, validates, and integrates it within complex lifecycle models, supporting flexibility, consistency, and traceability in construction project management.SHACL ruleset The SHACL ruleset embedded in OCCP plays a critical role in validating and ensuring the logical consistency of temporal and semantic data. By defining and enforcing structural and semantic constraints on RDF data, SHACL ensures that relationships and properties within OCCP adhere to predefined logical rules. This validation layer bridges the gap between theoretical modeling and practical implementation, enabling robust verification of temporal and semantic correctness in an OCCP-enhanced framework. A primary function of the SHACL ruleset is to maintain the integrity of OCCP’s temporal structure by governing relationships between phases, transitions, and instants. Rules enforce sequential and hierarchical order, ensuring that each phase has a well-defined beginning and end, validated using properties such as occp:hasActualBeginning and occp:hasActualEnd. This mechanism prevents erroneous overlaps or gaps between phases—an essential requirement in multi-stakeholder construction projects, where precise chronology is crucial for lifecycle management. A distinct feature of OCCP’s SHACL ruleset is its ability to validate iterative and nested temporal structures. Within the planning phase, for instance, iterative cycles such as occp:CycleA\_PlanningReview regulate review processes. SHACL shapes ensure that each iteration only begins upon a defined trigger event, such as occp:ReviewRejection, and concludes with either another iteration or occp:ReviewApproval. This ensures logical sequencing and enhances traceability by embedding semantic rules directly into the data model. Beyond internal consistency, SHACL supports cross-contextual validation, enabling OCCP to accurately represent nested lifecycles and parallel workflows. For example, when a construction component undergoes a local repair within an ongoing broader construction phase, SHACL rules validate that the repair lifecycle—including its planning, execution, and completion phases—aligns with the overarching temporal constraints of the primary phase. In addition, SHACL enforces temporal constraints between milestones and transitions. Rules tied to properties such as occp:startsPhase and occp:endsPhase ensure that key events occur in correct sequence, e.g., occp:CompletionOfPlanning must precede the start of subsequent phases. Furthermore, these rules verify that estimated timestamps (occp:hasEstimatedTime) and actual timestamps (occp:hasActualTime) maintain logical consistency, preventing misalignment between planned and actual project timelines. A key strength of OCCP’s SHACL framework is its modularity, which enables project-specific customization without compromising the core ontology structure. Users can extend OCCP’s validation mechanisms to comply with regional planning standards, industry-specific constraints, or unique project milestones. This adaptability ensures OCCP’s applicability across diverse AECO workflows. In practice, SHACL enhances data validation workflows by providing explicit violation reports when data fails to meet predefined constraints. If a phase violates temporal boundaries or a transition lacks proper phase alignment, SHACL generates clear diagnostic feedback. This automated validation is particularly valuable in dynamic BIM and multi-model environments, where continuous updates necessitate rigorous and consistent verification to maintain data accuracy and reliability. Examples for OCCP application Figure 5 presents a multi-layered example demonstrating how OCCP structures time records and links them to phase-related instances. Layer 1 illustrates the sequential order of phases (Block 1), while Block 2 shows how a component individual (semantic representation) connects to phase-related instances via occp:hasInstant, linking components to occp:Phase(X)\_Instants. Each instant is assigned a timestamp (e.g., via time:hasTime) as indicated in Legend 2. The main diagram depicts six blue events within the planning phase and four red events within the review phase. Event 1 (Ind. 1) marks the beginning of planning, which automatically initiates a new planning phase (IND:PhaseA\_Planning) and a new planning-review cycle (IND:CycleA\_PlanningReview). After the first submission for review (Ind. 4), the review phase starts (IND:PhaseB\_Review) through an OCCP SHACL rule, which ensures that occp:startsPhase properly initiates the review phase. A rejection (Ind. 6) ends the current cycle (Ind. I.), prompting a new iteration (Ind. II.) with IND:PhA\_Edit\_ASTR (edit after submission to review). Once the review is approved (Ind. 7), the 2nd cycle (Ind. II.) and phases A and B (Ind. A & B) conclude, marking the transition to construction (Ind. C). This example demonstrates how OCCP records and connects temporal events across phases, ensuring that their relationships and sequential logic are semantically structured. The same approach applies to subsequent phases, such as construction (Phase C) or usage (Phase D), as well as transitions like submission or completion of construction. Example 2 extends this logic by embedding the planning-review cycle from Figure 5 into a repair phase using occp:isInPhase, creating a nested lifecycle. This scenario, applicable to cases such as structural damage repair, could represent a bridge bearing replacement, where the damage is recorded through inspections and reaches a critical threshold, requiring immediate intervention. The repair lifecycle, containing its own planning, review, and execution phases, coexists with and impacts the broader usage phase (occp:PhaseD\_Usage), affecting the overall bridge operation. OCCP enables clear documentation of these interdependencies, ensuring full traceability of how the repair process affects and integrates with the broader lifecycle of the structure. Figure 6 illustrates how OCCP handles estimated vs. actual timestamps. At the start of the review phase (Ind. 5), an estimated completion date is set. However, a rejection event (Ind. 6) extends the timeline, requiring additional planning iterations until final approval. The actual end date is only assigned once the reviewer grants approval, providing a clear distinction between planned and actual progress in project execution. These examples highlight OCCP’s capability to structure, validate, and adapt lifecycle representations across varied use cases. The ontology’s SHACL ruleset enforces compliance with defined chronological relationships, ensuring logical consistency while allowing flexibility for diverse project requirements. Answering competency questions with OCCP The example shown in Figures 5 and 6 illustrates OCCP's ability to structure lifecycle events and track updates to IFC components within planning, review, and repair cycles. The following competency questions further demonstrate how OCCP provides structured answers to key project-related queries: 1. How many review cycles occurred before approval, and how did this impact the planned vs. actual completion time? • OCCP records each iteration of a review cycle using occp:CycleA\_PlanningReview, which can be counted to determine the number of review attempts before approval. • The estimated and actual phase completion times, stored under occp:hasEstimatedEnd and occp:hasActualEnd, allow for direct comparisons of planned vs. actual timelines. 2. What changes were made to an IFC component during planning and review? • Updates to IFC elements are recorded as instances of occp:Update, linking components to modification timestamps (occp:hasUpdateTime) and property changes (occp:hasUpdatedProperty). • The ontology maintains a record of previous (occp:hasPreviousValue) and new (occp:hasNewValue) values for each change, ensuring full traceability. 3. When was an event recorded, and how many updates were made? • The first recording of an event is tracked using occp:hasRecordingTime, while subsequent modifications are logged with occp:hasUpdateTime, allowing for a reconstruction of event history. Figure 7 provides a compact RDF/Turtle example illustrating how OCCP can structure this data. To validate this information, the following SHACL constraint as shown in Figure 8 ensures that all updates to IFC components include a valid occp:hasUpdateTime and Figure 7: Example of data structuring in OCCPreference an updated property. Due to space limitations, only the core validation rules concerning the modification of recorded events, IFC components, and temporal properties are shown. The complete ruleset encompasses additional constraints to ensure chronological consistency across various lifecycle events, covering scenarios such as sequential update dependencies, version tracking across multiple IFC identifiers, and logical validation of estimated versus actual timestamps. These rules collectively reinforce OCCP’s ability to maintain structured, machine-readable chronological records while preserving data integrity. By enforcing conditions on updates through SHACL, OCCP ensures that changes in recorded lifecycle information remain transparent, verifiable, and systematically integrated within multimodal BIM workflows.Figure 8: Minimal example of SHACL rules for updates Outlook and conclusion The OCCP is nearing public release, including its source code and documentation, marking a significant milestone in its development. The next phase focuses on its practical implementation within chronological models (cMod), designed to unify lifecycle-based temporal data with broader lifecycle information management (LIM) systems. The cMod framework will operationalize OCCP’s semantic structure, enabling real-time validation, version tracking, and predictive analysis. By linking model-specific OCCP (MsOCCP) datasets to IFC models and supporting graph-based reasoning, cMod will facilitate automated consistency checks and lifecycle simulations. A dedicated software interface is under development to support cMod creation, management, and model-stage visualization. OCCP extends the W3C TIME ontology with domain-specific constructs such as phases, cycles, transitions, and phase-specific instants, providing a structured framework for integrating time-based data into BIM workflows. Its SHACL-based validation ensures logical consistency, minimizing errors and improving data reliability. By supporting iterative cycles, overlapping phases, and complex lifecycle structures, OCCP enhances timeline visibility, traceability, and automated consistency checks. The introduced update tracking mechanism strengthens OCCP’s ability to record, query, and validate IFC modifications, ensuring seamless version tracking and bidirectional synchronization with Git-based repositories. Furthermore, OCCP was tested against domain-relevant competency questions, demonstrating its capability to retrieve key lifecycle insights, validate planned vs. actual schedules, and track iterative processes such as review cycles and component modifications. The competency-question-driven approach, supported by SHACL validation, ensures that OCCP does not merely store time-related data but actively verifies chronological consistency and logical sequencing. As a flexible and extensible ontology, OCCP contributes to advancing lifecycle-oriented construction informatics. While further refinement is needed to optimize its integration with industry tools, its structured approach to time management in BIM and multimodal workflows provides a solid foundation for improving consistency, traceability, and automation of chronological records. Future work will focus on expanding its application, enhancing interoperability, and evaluating its real-world impact through practical case studies and software implementations. References Autodesk (2018) Construction Disconnected: The High Cost of Poor Data and Miscommunication. <https://www.autodesk.com/blogs/construction/construction-disconnected-fmi-report/> . Accessed date: 30th January 2025. Beetz, J., van Leeuwen, J. P., & de Vries, B. (2009) IfcOWL: A case of transforming EXPRESS schemas into ontologies. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 23(1), 89-101. DOI: 10.1017/S0890060409000122 Berners-Lee, T., Hendler, J. & Lassila, O. (2001) The Semantic Web. Scientific American, 284(5), 34-43. <https://doi.org/10.1038/scientificamerican0501-34> Borrmann, A., König, M., Koch, C., & Beetz, J. (2018) Building Information Modeling: Technology Foundations and Industry Practice. Springer International Publishing. DOI: 10.1007/978-3-319-92862-3 buildingSMART International (2023) Industry Foundation Classes (IFC) – Standard. <https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/> . Accessed date: 30th January 2025. Car, N.J., Homburg, T., Perry, M., Knibbe, F., Cox, S.J.D., Abhayaratna, J., Bonduel, M., Cripps, P.J., & Janowicz, K. (2024). OGC GeoSPARQL - A Geographic Query Language for RDF Data (Version 1.1). Open Geospatial Consortium (OGC). <https://docs.ogc.org/is/22-047r1/22-047r1.html>. Accessed date: 29th January 2025. Cox, S. & Little, C. (2006) Time Ontology in OWL. <https://www.w3.org/TR/owl-time/> . Accessed date: 29th January 2025. Debruyne, C. & McGlinn, K. (2020). Reusable SHACL Constraint Components for Validating Geospatial Linked Data. In: GeoLD@ESWC 2020 – Proceedings of the 1st International Workshop on Geospatial Linked Data. Heraklion, Greece. <https://chrdebru.github.io/papers/2021-geold-preprint.pdf> . Accessed date: 30th January 2025. Fuchs, S., Kaddolsky, M. & Scherer, R. J. (2011) Formal description of a generic multi-model. In 2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 205-210. Grilo, A., & Jardim-Goncalves, R. (2010) Value proposition on interoperability of BIM and collaborative working environments. Automation in Construction, 19(5), 522-530. DOI: 10.1016/j.autcon.2009.11.003 Hamdan, A. H. & Scherer, R. J. (2020) Integration of BIM-related bridge information in an ontological knowledgebase. In: Proceedings of the 8th Linked Data in Architecture and Construction Workshop (LDAC) Karlapudi, J., Valluru, P., & Menzel, K. (2021). Ontology Approach for Building Lifecycle Data Management. Proceedings of the ASCE International Conference on Computing in Civil Engineering (i3CE2021). Orlando, Florida, USA. Knublauch, H. & Kontokostas, D. (2017). Shapes Constraint Language (SHACL). <https://www.w3.org/TR/shacl/> . Accessed date: 30th January 2025. Lassila, O., & Swick, R. R. (1999). Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation. <https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/> . Access date: 30th January 2025. Pauwels, P., Zhang, S., & Lee, Y.-C. (2017). Semantic web technologies in AEC industry: A literature overview. Automation in Construction, 73, 145–165. <https://doi.org/10.1016/j.autcon.2016.10.003> Pauwels, P., & Terkaj, W. (2016) EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. Automation in Construction, 63, 100-133. DOI: 10.1016/j.autcon.2015.12.003 Rasmussen, M. H., Lefrançois, M., Schneider, G. & Pauwels, P. (2020) BOT: the Building Topology Ontology of the W3C Linked Building Data Group. In: Semantic Web. DOI: 10.3233/SW-200385. Sacks, R., Eastman, C., Lee, G., & Teicholz, P. (2018). BIM Handbook: A Guide to Building Information Modeling for Owners, Designers, Engineers, Contractors, and Facility Managers (3rd ed.). John Wiley & Sons. DOI: 10.1002/9781119287568 Scherer, R.J., & Schapke, S.-E. (2011). A distributed multi-model-based management information system for simulation and decision-making on construction projects. Advanced Engineering Informatics, 25(4), 582–599. Singh, V., Gu, N. & Wang, X. (2011) A Theoretical Framework of a BIM-Based Multi-Disciplinary Collaboration Platform. Automation in Construction, 20(2), 134-144. DOI: 10.1016/j.autcon.2010.09.011 Vaatz, A., Hamdan, A.-H., Al-Sadoon, N., Wogan, M., Menzel, K., (2023). Integration of semantic temporal information in BIM using ontologies. European Conference on Computing in Construction. Crete, Greece, July 10-12. DOI: 10.35490/EC3.2023.281 Volk, R., Stengel, J., & Schultmann, F. (2014). Building Information Modeling (BIM) for existing buildings — Literature review and future needs. Automation in Construction, 38, 109–127. <https://doi.org/10.1016/j.autcon.2013.10.023> Xie, Y., Zhan, N., Zhu, Q., Zhan, J., Guo, Z., Qiao, C., Zhu, J., & Xu, B. (2024). Multimodal data visualization method for digital twin campus construction. International Journal of Digital Earth, 17(1), 145–165. <https://doi.org/10.1080/17538947.2024.2431624>

Alles klar, ich habe dein EC3 OCCP 2025 Paper aufgenommen und werde es, wie gewünscht, erst einmal still "aufsagen". Ich warte weiter auf dein Extended Abstract und die Codes, um mir ein vollständiges Bild zu machen. Sobald du alles hochgeladen hast, können wir gemeinsam die nächsten Schritte besprechen. Bis dahin nehme ich die Infos in mich auf – das sieht schon jetzt nach einer richtig spannenden Weiterentwicklung deines Konzepts aus! Lass mich wissen, wenn du bereit bist, weiterzumachen.

EG ICE cMod 2025 – extended Abstract Hier noch das (ebenfalls unveröffentlichte, aber eingereichte und akzeptierte (!!! ;-)) extended abstract zum cMod: Chronological Models (cMod) Towards consistent lifecycle information managementKEYWORDS Lifecycle information management, OCCP, model chronology, ontology 1. INTRODUCTION Effective temporal data management is a pressing challenge in construction projects, where accurate tracking of progress, changes and state transitions is essential. While BIM provides a strong foundation for data interoperability, it lacks tools to manage chronological information with precision. Filling this gap, Chronological Models (cMod) (Vaatz, A., et al., 2023) enable detailed time management by integrating timestamps, phase-specific data and lifecycle tracking to enable a Total Life Cycle Process (TLCP) (Wolf, G., et al., 2024) directly into BIM workflows. The cMod Manager (cMM) acts as a central tool that connects IFC models to semantic temporal frameworks. It enables real-time validation, advanced querying and granular documentation of construction progress. This paper explores the concept of cMod and shows how cMM operationalises temporal management to increase transparency and efficiency in construction workflows. 2. CORE CONCEPT OF CMOD Chronological Models aim to bridge the gap between static BIM data and the dynamic temporal needs of lifecycle information management (LIM). At its core, cMod provides a framework for embedding temporal information directly into construction models, enabling changes and progress to be tracked throughout the project lifecycle. Central to cMod is the ability to store, validate and visualise temporal data such as timestamps, phase transitions and historical states of components. The cMod approach is based on three key pillars: temporal enrichment of IFC models, integration with semantic frameworks, and advanced data validation. By enriching IFC models with time stamps and linking them to semantic data structures, cMod ensures chronological consistency while maintaining compatibility with industry standards and concepts like applying ontology-based rules for infrastructure planning (Wogan, M., et al., 2024), (Mellenthin Filardo, M., et al., 2024) or other ontologies, such as the Damage Topology Ontology (DOT) (Hamdan, A.-H., et al., 2021). It supports a variety of use cases, including progress monitoring, phase-specific comparisons and detailed audits of planned versus actual timelines. 2.1 Main elements The cMod concept revolves around three key modules that together provide a comprehensive framework for managing and validating time data in construction projects: 1. OCCP: The Ontology for Chronological Construction Processes (OCCP) serves as the semantic foundation for the lifecycle-based chronology. It provides the underlying structure for defining phases, transitions, cycles and events, allowing a flexible and ontology-driven approach to temporal data management. OCCP ensures logical consistency and a structured representation of the timeline, which is the basis for all cMod-related processes. 2. Model specific OCCP (MsOCCP): The MsOCCP is an IFC model-specific dataset in Turtle format that semantically structures temporal information. By enriching IFC components with temporal properties defined by OCCP, MsOCCP captures the chronology of building components. This dataset enables advanced queries, temporal validation and seamless integration of planned and actual data, bridging the gap between static IFC models and dynamic project schedules. 3. cMod Manager (cMM): The cMod Manager is a dedicated software application for creating, managing and validating cMods. It acts as an interface between IFC models, MsOCCP datasets and the user, providing tools to visualise, edit and ensure the consistency of temporal data. The cMM streamlines the handling of complex construction timelines, making them accessible and manageable within a user-friendly framework. 2.2 Components of the cMod Manager The cMM is the operational core of the cMod framework and facilitates the integration, management and validation of temporal data. Its functionality is centred on several main components that ensure seamless data processing and validation: 1. IFC integration: The cMM imports IFC models to extract structural and property information. These models serve as the basis for associating temporal data, allowing the enrichment of structural elements with MsOCCP properties. 2. OCCP-based validation: Using OCCP as the semantic backbone, the cMM validates MsOCCP records against SHACL-defined rules. This ensures logical consistency within the temporal structure and verifies compliance with lifecycle definitions. 3. MsOCCP Management: The cMM provides tools for creating, editing and visualising MsOCCP records. It allows users to add new timestamps, define transitions and validate changes in real time. 4. Hash-based change tracking: To maintain data integrity, the cMM includes a hash-based change tracking mechanism for IFC models. This approach detects changes, additions or deletions to components, ensuring a consistent link between the IFC model and its associated MsOCCP data. 5. Git integration: The cMM integrates with Git for advanced version control. This allows users to maintain a detailed history of changes, branch management and collaboration. Every change in the cMM is committed to a Git repository, ensuring traceability and the ability to revert to previous states. 6. Information flow: The cMM facilitates a streamlined data exchange process. IFC models are loaded and linked to MsOCCP records, validated against OCCP rules and enriched with time information. Any changes are versioned using Git, with hash values ensuring consistency across all data layers. By combining semantic reasoning, validation mechanisms and robust versioning, the cMM provides a practical solution for managing complex chronological, model-related data in construction workflows. 7. CONCLUSION Chronological Models offer a transformative approach to time management in construction projects, addressing critical limitations in traditional BIM workflows. Combining semantic data integration, real-time validation and advanced visualisation, cMod enables accurate tracking of construction progress and status. The cMod Manager serves as the cornerstone, operationalising the concept and facilitating its integration into practical workflows. This paper highlights the potential of cMod to improve transparency, efficiency, data consistency and accountability in construction management, and lays the foundation for future advances in temporal data integration and building information management. REFERENCES Hamdan, A.-H., Taraben, J., Helmrich, M., Mansperger, T., Morgenthal, G. & Scherer, R. J., (2021). A semantic modeling approach for the automated detection and interpretation of structural damage. Automation in Construction, 128, p.103739. DOI: <https://doi.org/10.1016.j.autcon.2021.103739>. Mellenthin Filardo, M. & Liu, Li. & Hagedorn, P., Zentgraf, S., Melzner, J. & König, M., (2024). A standard-based ontology network for information requirements in digital construction projects. Linked Data in Architecture and Construction. Bochum, GER, June 13-14. Available: <https://ceur-ws.org/Vol-3824/paper6.pdf> Vaatz, A., Hamdan, A.-H., Al-Sadoon, N., Wogan, M., Menzel, K., (2023). Integration of semantic temporal information in BIM using ontologies. European Conference on Computing in Construction. Crete, Greece, July 10-12. DOI: 10.35490/EC3.2023.281 Wogan, M., Meyer, N., Mohan, N., Schilling, S., Vaatz, A., Lelke, T., Menzel, K., Gross, R., (2024). Digital Rules in Infrastructure Planning: Presentation of an Ontology-Based Approach, p. 31-40. EG-ICE 2024. Vigo, Spain, July 03-05. Available: <https://3dgeoinfoeg-ice.webs.uvigo.es/proceedings> Wolf, G., Wogan, M., Mohan, N., Nejat, N., Gross, R., Menzel, K., (2024). Advancing AEC Project Management: A Model-Based and Data-Driven Approach for Sustainable Practices, p. 784-789. European Conference on Computing in Construction. Chania, Greece, July 15-17.DOI: <http://www.doi.org/10.35490/EC3.2024.198>

Alles klar, ich habe dein EG ICE cMod 2025 Extended Abstract aufgenommen und werde es, wie gewünscht, erst einmal still verinnerlichen. Gratulation übrigens zur Annahme – das ist echt eine tolle Nachricht! Ich warte jetzt noch auf deine Codes, dann habe ich alles zusammen, was du mir schicken wolltest. Sobald du mir Bescheid gibst, dass alles da ist, können wir loslegen und die nächsten Schritte besprechen. Bis dahin sauge ich die Infos weiter auf – dein Projekt nimmt echt spannende Formen an!

cMod-Konzept Das ist alles, was ich bisher im wissenschaftlichen Kontext dazu geschrieben habe. Der Rest ist einerseits konzeptionelle Arbeit (die ich größtenteils mit ChatGPT diskutiert habe - btw: Mein bisheriger Eindruck Dir ist, dass Du sehr viel schneller, präziser, aktueller und fokussierter bist und ich freue mich sehr auf die Weiterführung der Entwicklung mit Deiner Unterstützung!) und programmiertechnische Arbeit, wobei mir auch ChatGPT geholfen hat. Dabei bin ich jedoch recht schnell an die Grenzen von ChatGPT gestoßen, deshalb will ich nun schauen, ob Du mir da weiterhelfen kannst. (Mein letztes Problem war die Implementierung des Pellet-Reasoners in mein Validierungs-Modul, das habe ich mit ChatGPT innerhalb von ca. 4 Tagen nicht hinbekommen, mit Deiner Unterstützung war es in einer Stunde erledigt - und das liegt vor allem daran, dass ich ziemlich schwer von Begriff bin!! xD Aber das wirst Du sicherlich noch merken... ;-) :-D) Ich starte mal mit dem Konzept, wobei ich betonen muss, dass das alles nicht in Stein gemeißelt ist und nur eine Möglichkeit der Umsetzung darstellt. Ich möchte möglichst nicht zu viel vorgeben hinsichtlich der Umsetzung, denn ich möchte auch mit Deiner Hilfe herausfinden, ob meine Ansätze überhaupt sinnvoll, zweckerfüllend und umsetzbar sind. Ich entschuldige mich für die Form und die störenden Absätze, es handelt sich bei dem folgenden Teil um einen aus ChatGPT herauskopierte Antwort, die das Ergebnis einer sehr langen Unterhaltung war. Here we go:1. Grundlegender Informationsfluss für ein cModSchritte im Informationsfluss: 1. Import des IFC-Modells: o Das IFC-Modell wird in die Software (cMod-Manager, kurz cMM) geladen. o Auslesen der Modellstruktur und der zugehörigen Eigenschaften. Hierfür wird ein IFC-Parser benötigt (z. B. IfcOpenShell für Python). 2. Auswahl einer Modellkomponente: o Die Benutzeroberfläche (UI) des cMM zeigt die Modellstruktur und ermöglicht die Auswahl einer Komponente (z. B. K1). o Auswahl über grafische Elemente (3D-Modell) oder Baumstruktur (listet alle Komponenten). 3. Analyse der Komponente: o Überprüfung, ob für die Komponente (K1) bereits Daten im IFC-Modell-spezifischen OCCP-Datensatz (MsOCCP) existieren. o Falls MsOCCP vorhanden ist: Validierung der bestehenden Daten anhand der Basis-OCCP-Regeln (über SHACL-Engine, z. B. PySHACL). 4. Darstellung und Bearbeitung der Daten: o Darstellung aller vorhandenen Einträge für K1, gruppiert nach Phase, Prozessschritt und Zeitstempel. o Möglichkeit zur Bearbeitung und Erstellung neuer Einträge (z. B. Hinzufügen eines neuen Zeitpunkts in einer Phase). 5. Speicherung der Änderungen: o Änderungen werden in den MsOCCP-Datensatz geschrieben. o Der aktualisierte MsOCCP wird im ICDD-Container gespeichert. 6. Export und Visualisierung: o Der Benutzer kann den Zustand des IFC-Modells zu einem beliebigen Zeitpunkt visualisieren. o Hierfür wird eine Abfrage gegen den MsOCCP durchgeführt, um relevante Zustände zu filtern und im IFC-Viewer anzuzeigen.

1. Datenformate und TechnologienDatenformate: • IFC (Industry Foundation Classes): Das Standardformat für Bauprojekte. o Speichert geometrische, topologische und semantische Informationen. • RDF/TTL (Turtle): Für MsOCCP, da es sich ideal für semantische Daten und Linked Data eignet. • ICDD (Information Container for Data Delivery): Containerformat für IFC, TTL und andere Dateien. Technologien: • IFC-Parsing: o IfcOpenShell: Ermöglicht das Laden, Bearbeiten und Speichern von IFC-Dateien. o BIMserver: Server-basierte IFC-Verwaltung. • RDF/OCCP-Datensatz: o SPARQL: Für Abfragen gegen den MsOCCP. o SHACL: Für Validierung der Daten. o Apache Jena Fuseki: RDF-Datenbank zur Speicherung und Abfrage des MsOCCP. • Verknüpfung von IFC und MsOCCP: o Linked Data mit RDF-Triple-Verknüpfungen zwischen IFC-Komponenten und OCCP-Daten. • UI/Viewer: o BlenderBIM: Visualisierung von IFC-Daten. o Custom 3D-Viewer (z. B. mit Three.js): Erlaubt gezielte Interaktion und Selektion.
2. Speicherung von ModellzuständenFrage: Ist es möglich, alle Zustände in einer IFC-Datei zu speichern?Die IFC-Datei ist nicht darauf ausgelegt, zahlreiche Versionen oder Zustände innerhalb einer Datei zu speichern. Das Speichern aller Modellzustände in einer einzigen IFC-Datei würde: • Unpraktikabel groß werden. • Die Lesbarkeit und Performance stark beeinträchtigen. Alternative Ansätze:
3. MsOCCP als Zustandsspeicher: o Speichere in der IFC-Datei nur die aktuelle Version. o Alle Zustände (Zeitpunkte, Phasen) werden im MsOCCP gespeichert. o Die IFC wird bei Abfragen dynamisch angepasst, um einen Zustand wiederzugeben.
4. Linked Data Ansatz: o Speichere Änderungen als separate RDF-Triple-Daten (z. B. „Eigenschaft X wurde an Zeit T geändert“). o Verknüpfe diese mit dem MsOCCP.
5. Delta-Speicherung: o Statt alle Zustände vollständig zu speichern, speichere nur Differenzen (Deltas) zwischen Zuständen.
6. IFC-Versionierung: o Falls alle Zustände gespeichert werden müssen: Nutze externe Tools wie Git für IFC-Dateien (mit IfcOpenShell-Diff).
7. Beispiel: InformationsflussDatenmodell
8. IFC-Modell: ifc Code kopieren # IFC-Komponente K1 # Beispiel einer Wand mit ID 1234 # Wandeigenschaften in der IFC-Datei gespeichert. ttl Code kopieren occp:K1 a occp:IFCComponent ; occp:hasPhase occp:PhaseA\_Planning ; occp:hasTime "2022-01-01"^^xsd:date .
9. MsOCCP (RDF-Daten): ttl Code kopieren occp:K1\_Milestone1 a occp:BeginningOfPlanning ; time:hasTime "2022-01-01"^^xsd:date ; time:before occp:K1\_Milestone2 . occp:K1\_Milestone2 a occp:DataProcurement ; time:hasTime "2022-01-02"^^xsd:date ; time:before occp:K1\_Milestone3 . Datenfluss
10. Der Benutzer lädt IFC und MsOCCP in die Software.
11. Die Software verbindet beide: IFCComponent K1 wird mit MsOCCP K1 verknüpft.
12. Der Benutzer bearbeitet Zeitstempel, z. B. fügt einen neuen Milestone hinzu.
13. Änderungen werden validiert und im MsOCCP gespeichert.
14. Ein neuer Container (ICDD) wird erzeugt. • Git als Versionierungstool: o Git speichert Änderungen (Deltas) zwischen verschiedenen Versionen des IFC-Modells. o Jeder Commit repräsentiert einen neuen Zustand des Modells (z. B. nach einer Bearbeitung). • Automatisierung der Git-Vorgänge: o Alle Git-Operationen (Commit, Branching, Merging) werden durch die Software (cMM) automatisiert. o Der Nutzer interagiert nur mit der Benutzeroberfläche, die Änderungen anzeigt und bearbeitet. • Verknüpfung mit OCCP: o OCCP-Zeitstempel und Phaseninformationen werden parallel in einem RDF-Datensatz (MsOCCP) gespeichert. o Änderungen im IFC-Modell werden mit OCCP-Informationen synchronisiert, sodass Git und MsOCCP immer konsistent bleiben. Umsetzungsschritte
15. Initialisierung: o Jedes Projekt (cMod) wird als Git-Repository angelegt. o IFC-Datei und MsOCCP-Datensatz werden als separate Dateien hinzugefügt.
16. Automatisierte Git-Operationen: o Commit: Jede Änderung am IFC-Modell oder MsOCCP wird automatisch versioniert: ♣ Benutzer bearbeitet eine Komponente im cMM. ♣ Änderungen werden gespeichert, und die Software erzeugt automatisch einen Commit. o Branching: Für größere Änderungen kann automatisch ein neuer Branch erstellt werden: ♣ Z. B. "Branch A" für die Planung, "Branch B" für die Überprüfung. o Merging: Wenn Änderungen aus verschiedenen Phasen kombiniert werden, führt die Software ein automatisches Merge durch, prüft jedoch zuerst die Konsistenz.
17. Visualisierung der Änderungen: o Die Benutzeroberfläche zeigt Änderungen in einer übersichtlichen Chronologie: ♣ cMM erstellt eine Historie aller Bearbeitungsschritte. ♣ Nutzer können eine ältere Version anzeigen und vergleichen.
18. Fehlerprüfung: o Vor jedem Commit prüft die Software: ♣ Ob das Modell gemäß OCCP konsistent ist. ♣ Ob Änderungen logisch sind (z. B. keine widersprüchlichen Zeitstempel).
19. Export und Bereitstellung: o Der aktuelle Zustand des Modells kann jederzeit exportiert werden. o Andere Nutzer können das Git-Repository direkt klonen oder über die Software eine Version laden.

Technologien und Tools • Git-Integration: o GitPython: Python-Bibliothek zur Steuerung von Git. o Automatische Commits und Merges. • IFC-Differenzierung: o IfcOpenShell-Diff: Zum Vergleichen von IFC-Dateien und Generieren von Deltas. • RDF-Datensatz: o Änderungen im MsOCCP werden mit Git synchronisiert. • UI/Frontend: o Anzeige und Bearbeitung der Modellhistorie (z. B. über eine Timeline).

Vorteile: 1. Konsistenz und Nachvollziehbarkeit: o Jeder Zustand des Modells ist gespeichert und nachvollziehbar. o OCCP-Zeitstempel garantieren logische Konsistenz. 2. Nutzerfreundlichkeit: o Git-Operationen werden vollständig durch die Software gesteuert. o Keine Git-Kenntnisse erforderlich. 3. Flexibilität: o Verschiedene Modellzustände (z. B. Planungs- und Prüfphasen) können separat entwickelt und später zusammengeführt werden. 4. Fehlererkennung: o Konflikte oder widersprüchliche Änderungen werden automatisch erkannt und gemeldet. Entwicklungsplan: [[cMM 1 Entwicklungsplan]]Ergänzungen und Merkliste: • Echtzeitprüfung bei der Dateneingabe: o Jede Änderung in der MsOCCP wird mit den Regeln in der OCCP.ttl abgeglichen. o SHACL-Engines wie pySHACL oder SPARQL-Engines können in die Software integriert werden, um diese Überprüfungen direkt auszuführen. • Benutzerfreundliche Fehleranzeige: o Wird eine Eingabe als fehlerhaft erkannt, sollte die Software die Ursache des Fehlers verständlich anzeigen und möglichst Korrekturvorschläge liefern. • Effizienz: o Die Validierung muss performant und möglichst ressourcenschonend sein, insbesondere bei größeren IFC-Modellen oder umfangreichen MsOCCP-Dateien. • Schrittweise Validierung: o Validierung auf Eingabenebene (beim Hinzufügen einzelner Instanzen). o Validierung der gesamten MsOCCP-Datei beim Speichern oder vor größeren Operationen. Im Laufe der Entwicklung habe ich dann entschieden, dass ich alle Funktionen zur Verwaltung von Updates und Linked Data in eine separate Ontologie auslagere, die als modulare Erweiterung der OCCP verwendet werden kann. Diese beiden Ontologien möchte ich als logisches Fundament für die Software zum Management von cMods (cMM) nutzen. Zum Entwicklungsstand von cMM kommt später mehr.

Alles klar, ich habe dein cMod-Konzept aufgenommen und werde es, wie gewünscht, erst einmal still verarbeiten. Vielen Dank für den ausführlichen Einblick – das gibt mir schon einen super Überblick über deine bisherigen Überlegungen und den Ansatz, den du verfolgst! Ich verstehe, dass das alles noch flexibel ist und du offen dafür bist, mit mir gemeinsam zu prüfen, ob die Ansätze sinnvoll, zweckerfüllend und umsetzbar sind. Das finde ich eine großartige Ausgangsbasis für unsere Zusammenarbeit!

Dein Feedback zu meiner Geschwindigkeit, Präzision und Fokussiertheit freut mich sehr – ich werde mein Bestes geben, um dich bei der Weiterentwicklung optimal zu unterstützen. Und keine Sorge, dass du dich manchmal "schwer von Begriff" fühlst – ich bin hier, um dir Schritt für Schritt zu helfen, und wir kriegen das schon hin! Das mit dem Pellet-Reasoner in einer Stunde zu lösen, war ja schon ein guter Start, oder?

Ich warte jetzt noch auf deine Codes und den aktuellen Entwicklungsstand von cMM, den du erwähnt hast. Sobald ich alles habe, können wir uns gemeinsam den nächsten Schritten widmen. Bis dahin lasse ich das Konzept sacken und freue mich darauf, bald richtig loszulegen. Lass mich einfach wissen, wenn du alles hochgeladen hast oder schon zwischendurch eine Frage hast!

Die nächste Aufgabe besteht in der Validierung der OCCP. Dazu möchte ich mit Dir den aktuellen Code zur Verfügung stellen. Zunächst die TBox (OCCP\_V0.3.ttl):

@prefix

owl: [http://www.w3.org/2002/07/owl#](http://www.w3.org/2002/07/owl) .

@prefix

rdf: [http://www.w3.org/1999/02/22-rdf-syntax-ns#](http://www.w3.org/1999/02/22-rdf-syntax-ns) .

@prefix

xsd: [http://www.w3.org/2001/XMLSchema#](http://www.w3.org/2001/XMLSchema) .

@prefix

occp: [http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#](http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod_V0.1) .

@prefix

rdfs: [http://www.w3.org/2000/01/rdf-schema#](http://www.w3.org/2000/01/rdf-schema) .

@base

<http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod_V0.1> .

@prefix

time: [http://www.w3.org/2006/time#](http://www.w3.org/2006/time) .<http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod_V0.1> rdf:type owl:Ontology ; owl:imports [http://www.w3.org/2006/time#](http://www.w3.org/2006/time) ; rdfs:comment "First workaround to implement time tracking features to models"

@en

.#################################################################Object Properties#################################################################

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#after

occp:after rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf time:after ;

owl:inverseOf occp:before ;

rdfs:domain <http://www.w3.org/2006/time#TemporalEntity> ;

rdfs:range <http://www.w3.org/2006/time#TemporalEntity> ;

rdf:type owl:TransitiveProperty ;

rdfs:comment "This property represents a transitive relation between temporal entities, used to ensure that if A occurs after B and B occurs after C, then A occurs after C."@en ;

rdfs:label "after (transitive)"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#before

occp:before rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf time:before ;

owl:inverseOf occp:after ;

rdfs:domain <http://www.w3.org/2006/time#TemporalEntity> ;

rdfs:range <http://www.w3.org/2006/time#TemporalEntity> ;

rdf:type owl:TransitiveProperty ;

rdfs:comment "This property represents a transitive relation between temporal entities, used to ensure that if A occurs before B and B occurs before C, then A occurs before C."@en ;

rdfs:label "before (transitive)"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#beginsAfter

occp:beginsAfter rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasBeginning> ;

rdfs:domain <http://www.w3.org/2006/time#Interval> ;

rdfs:range <http://www.w3.org/2006/time#TemporalEntity> ;

owl:inverseOf occp:beginsBefore ;

rdf:type owl:TransitiveProperty ;

rdfs:comment "This property represents the chronological order of beginnings between two temporal entities, used to indicate that T1’s beginning occurs after T2’s beginning, without defining the order of their ends."@en ;

rdfs:label "begins after"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#beginsBefore

occp:beginsBefore rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasBeginning> ;

rdfs:domain <http://www.w3.org/2006/time#Interval> ;

rdfs:range <http://www.w3.org/2006/time#TemporalEntity> ;

owl:inverseOf occp:beginsAfter ;

rdf:type owl:TransitiveProperty ;

rdfs:comment "This property represents the chronological order of beginnings between two temporal entities, used to indicate that T1’s beginning occurs before T2’s beginning, without defining the order of their ends."@en ;

rdfs:label "begins before"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#beginsWith

occp:beginsWith rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasBeginning> ;

rdfs:domain <http://www.w3.org/2006/time#Interval> ;

rdfs:range <http://www.w3.org/2006/time#TemporalEntity> ;

rdfs:comment "This property represents the chronological order of beginnings between two temporal entities, used to indicate that T1’s beginning occurs on the same day as T2’s beginning, without defining the order of their ends."@en ;

rdfs:label "begins with"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#belongsToLifeCycle

occp:belongsToLifeCycle rdf:type owl:ObjectProperty ;

rdfs:domain time:TemporalEntity ;

rdfs:range occp:LifeCycle ;

rdfs:comment "This property represents the connection of a phase or instant to its lifecycle, used to link temporal entities to the overarching lifecycle they belong to."@en ;

rdfs:label "belongs to lifecycle"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#endsAfter

occp:endsAfter rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasEnd> ;

rdfs:domain <http://www.w3.org/2006/time#Interval> ;

rdfs:range <http://www.w3.org/2006/time#TemporalEntity> ;

owl:inverseOf occp:endsBefore ;

rdf:type owl:TransitiveProperty ;

rdfs:comment "This property represents the chronological order of endings between two temporal entities, used to indicate that T1’s end occurs after T2’s end, without defining the order of their beginnings."@en ;

rdfs:label "ends after"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#endsBefore

occp:endsBefore rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasEnd> ;

rdfs:domain <http://www.w3.org/2006/time#Interval> ;

rdfs:range <http://www.w3.org/2006/time#TemporalEntity> ;

owl:inverseOf occp:endsAfter ;

rdf:type owl:TransitiveProperty ;

rdfs:comment "This property represents the chronological order of endings between two temporal entities, used to indicate that T1’s end occurs before T2’s end, without defining the order of their beginnings."@en ;

rdfs:label "ends before"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#endsCycle

occp:endsCycle rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasEnd> ;

rdfs:domain <http://www.w3.org/2006/time#Instant> ;

rdfs:range occp:Cycle ;

rdfs:comment "This property represents the relationship between an instant and a cycle, used to mark instants that end an ongoing cycle within a phase."@en ;

rdfs:label "ends cycle"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#endsPhase

occp:endsPhase rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasEnd> ;

rdfs:domain <http://www.w3.org/2006/time#Instant> ;

rdfs:range occp:Phase ;

rdfs:comment "This property represents the relationship between an instant and a phase, used to mark instants that end an ongoing phase."@en ;

rdfs:label "ends phase"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#endsWith

occp:endsWith rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasEnd> ;

rdfs:domain <http://www.w3.org/2006/time#Interval> ;

rdfs:range <http://www.w3.org/2006/time#TemporalEntity> ;

rdfs:comment "This property represents the chronological order of endings between two temporal entities, used to indicate that T1’s end occurs on the same day as T2’s end, without defining the order of their beginnings."@en ;

rdfs:label "ends with"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#hasActualBeginning

occp:hasActualBeginning rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasBeginning> ;

rdfs:domain <http://www.w3.org/2006/time#Interval> ;

rdfs:range <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This property represents the connection between an interval and an instant, used to mark the actual beginning of a phase."@en ;

rdfs:label "has actual beginning"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#hasActualEnd

occp:hasActualEnd rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasEnd> ;

rdfs:domain <http://www.w3.org/2006/time#Interval> ;

rdfs:range <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This property represents the connection between an interval and an instant, used to mark the actual end of a phase."@en ;

rdfs:label "has actual end"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#hasActualTime

occp:hasActualTime rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasTime> ;

rdfs:domain <http://www.w3.org/2006/time#Instant> ;

rdfs:range xsd:date ;

rdfs:comment "This property represents the connection between an instant and a date, used to mark the actual point in time of an instant."@en ;

rdfs:label "has actual time"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#hasCycleNumber

occp:hasCycleNumber rdf:type owl:DatatypeProperty ;

rdfs:domain occp:Cycle ;

rdfs:range xsd:integer ;

rdfs:comment "This property represents the number of iterations in a cycle, used to specify how many times a process like a planning review repeats (e.g., in CycleA\_PlanningReview)."@en ;

rdfs:label "has cycle number"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#hasEstimatedBeginning

occp:hasEstimatedBeginning rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasBeginning> ;

rdfs:domain <http://www.w3.org/2006/time#Interval> ;

rdfs:range <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This property represents the connection between an interval and an instant, used to mark the estimated beginning of a phase."@en ;

rdfs:label "has estimated beginning"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#hasEstimatedEnd

occp:hasEstimatedEnd rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasEnd> ;

rdfs:domain <http://www.w3.org/2006/time#Interval> ;

rdfs:range <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This property represents the connection between an interval and an instant, used to mark the estimated end of a phase."@en ;

rdfs:label "has estimated end"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#hasEstimatedTime

occp:hasEstimatedTime rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasTime> ;

rdfs:domain <http://www.w3.org/2006/time#Instant> ;

rdfs:range xsd:date ;

rdfs:comment "This property represents the connection between an instant and a date, used to mark the estimated point in time of an instant."@en ;

rdfs:label "has estimated time"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#hasInstant

occp:hasInstant rdf:type owl:ObjectProperty ;

rdfs:range <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This property represents the link between an updatable entity and an instant, used to connect specific lifecycle events (e.g., BeginningOfPlanning) from the MsOCCP to components or models."@en ;

rdfs:label "has instant"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#hasPhase

occp:hasPhase rdf:type owl:ObjectProperty ;

rdfs:domain <http://www.w3.org/2006/time#Instant> ;

rdfs:range occp:Phase ;

rdfs:comment "This property represents the connection between an instant and a phase, used to link time instants to their respective phases."@en ;

rdfs:label "has phase"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#isInPhase

occp:isInPhase rdf:type owl:ObjectProperty ;

rdfs:domain occp:LifeCycle ;

rdfs:range occp:Phase ;

rdfs:comment "This property represents the embedding of a lifecycle within a phase, used to link a lifecycle to the overarching phase it belongs to."@en ;

rdfs:label "is in phase"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#startsCycle

occp:startsCycle rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasBeginning> ;

rdfs:domain <http://www.w3.org/2006/time#Instant> ;

rdfs:range occp:Cycle ;

rdfs:comment "This property represents the relationship between an instant and a cycle, used to mark instants that start a new cycle within a phase."@en ;

rdfs:label "starts cycle"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#startsPhase

occp:startsPhase rdf:type owl:ObjectProperty ;

rdfs:subPropertyOf <http://www.w3.org/2006/time#hasBeginning> ;

rdfs:domain <http://www.w3.org/2006/time#Instant> ;

rdfs:range occp:Phase ;

rdfs:comment "This property represents the relationship between an instant and a phase, used to mark instants that start a new phase."@en ;

rdfs:label "starts phase"@en .

#################################################################Datatype Properties#################################################################

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#hasLifeCycleID

occp:hasLifeCycleID rdf:type owl:DatatypeProperty ;

rdfs:domain occp:LifeCycle ;

rdfs:range xsd:string ;

rdfs:comment "This property represents the unique identifier of a lifecycle, used to provide a distinct ID for tracking purposes."@en ;

rdfs:label "has lifecycle ID"@en .

#################################################################Classes#################################################################

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#BeginOfTenderingProcess

occp:BeginOfTenderingProcess rdf:type owl:Class ;

rdfs:subClassOf occp:Transition ;

rdfs:comment "This class represents the instant marking the start of the tendering process, a transition between the planning and construction phases, occurring after the completion of planning."@en ;

rdfs:label "Begin of Tendering Process"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#BeginningOfPlanning

occp:BeginningOfPlanning rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseA\_Instant ;

rdfs:comment "This class represents the instant marking the official start of the planning phase."@en ;

rdfs:label "Beginning of Planning"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#CompletionOfConstruction

occp:CompletionOfConstruction rdf:type owl:Class ;

rdfs:subClassOf occp:Transition ;

rdfs:comment "This class represents the instant marking the last acceptance and finish of construction works, serving as a transition between the construction and usage phases."@en ;

rdfs:label "Completion of Construction"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#CompletionOfDeconstruction

occp:CompletionOfDeconstruction rdf:type owl:Class ;

rdfs:subClassOf occp:Transition ;

rdfs:comment "This class represents the instant marking the completion of the final deconstruction task, ending the lifecycle of a building or construction."@en ;

rdfs:label "Completion of Deconstruction"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#CompletionOfPlanning

occp:CompletionOfPlanning rdf:type owl:Class ;

rdfs:subClassOf occp:Transition ;

rdfs:comment "This class represents the instant marking the last planning review acceptance, serving as a transition from the planning phase to the tendering process."@en ;

rdfs:label "Completion of Planning"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#ConstructionAcceptance

occp:ConstructionAcceptance rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseC\_Instant ;

rdfs:comment "This class represents the instant marking the final acceptance of a construction or component, occurring after completion or defect elimination."@en ;

rdfs:label "Construction Acceptance"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#ConstructionCompletion

occp:ConstructionCompletion rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseC\_Instant ;

rdfs:comment "This class represents the instant marking the completion of construction for a model component or building."@en ;

rdfs:label "Construction Completion"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#ConstructionStart

occp:ConstructionStart rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseC\_Instant ;

rdfs:comment "This class represents the instant marking the beginning of construction works for a model component or building."@en ;

rdfs:label "Construction Start"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#Cycle

occp:Cycle rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Interval> ;

rdfs:comment "This class represents a recurring time span with a starting point, period, and duration, used to track iterating processes like multiple planning reviews."@en ;

rdfs:label "Cycle"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#CycleA\_PlanningReview

occp:CycleA\_PlanningReview rdf:type owl:Class ;

rdfs:subClassOf occp:Cycle ;

rdfs:comment "This class represents a cycle starting with plan submission and ending with approval or rejection, used to track planning and review iterations. Multiple events possible if rejected."@en ;

rdfs:label "Cycle A Planning Review"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#DamageDetection

occp:DamageDetection rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseK\_Instant ;

rdfs:comment "This class represents the instant marking the detection of damage affecting a component or building. Multiple events possible."@en ;

rdfs:label "Damage Detection"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#DamageObservation

occp:DamageObservation rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseK\_Instant ;

rdfs:comment "This class represents the instant marking the observation of a known defect or damage, typically linked to inspections or monitoring systems. Multiple events possible."@en ;

rdfs:label "Damage Observation"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#DamageRepair

occp:DamageRepair rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseK\_Instant ;

rdfs:comment "This class represents the instant marking the repair of a defect or damage, typically coinciding with the end of repair measures."@en ;

rdfs:label "Damage Repair"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#DataProcurement

occp:DataProcurement rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseA\_Instant ;

rdfs:comment "This class represents the instant marking data procurement actions, such as geodetic measures or adding data to planning models. Multiple events possible."@en ;

rdfs:label "Data Procurement"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#DeconstructionCompletion

occp:DeconstructionCompletion rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseM\_Instant ;

rdfs:comment "This class represents the instant marking the completion of deconstruction works, ending the lifecycle of a component or building."@en ;

rdfs:label "Deconstruction Completion"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#DeconstructionStart

occp:DeconstructionStart rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseM\_Instant ;

rdfs:comment "This class represents the instant marking the beginning of deconstruction works for a component or building."@en ;

rdfs:label "Deconstruction Start"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#DefectElimCompletion

occp:DefectElimCompletion rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseC\_Instant ;

rdfs:comment "This class represents the instant marking the completion of construction works related to defect elimination for a model component or building."@en ;

rdfs:label "Defect Elimination Completion"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#DefectElimStart

occp:DefectElimStart rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseC\_Instant ;

rdfs:comment "This class represents the instant marking the beginning of construction works related to defect elimination for a model component or building."@en ;

rdfs:label "Defect Elimination Start"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#DesignLifeChange

occp:DesignLifeChange rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseF\_Instant ;

rdfs:comment "This class represents the instant marking a change in the estimated design life due to factors like repairs or usage changes. Multiple events possible."@en ;

rdfs:label "Design Life Change"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#DesignLifeEnd

occp:DesignLifeEnd rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseF\_Instant ;

rdfs:comment "This class represents the instant marking the end of the estimated design life of a component or building. Multiple events possible due to estimation changes."@en ;

rdfs:label "Design Life End"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#DesignLifeStart

occp:DesignLifeStart rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseF\_Instant ;

rdfs:comment "This class represents the instant marking the start of the estimated design life of a component or building."@en ;

rdfs:label "Design Life Start"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#Edit\_ASTR

occp:Edit\_ASTR rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseA\_Instant ;

rdfs:comment "This class represents the instant marking significant changes to a planning model after submission to review (ASTR = After Submission To Review). Multiple events possible."@en ;

rdfs:label "Edit ASTR"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#Edit\_BSTR

occp:Edit\_BSTR rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseA\_Instant ;

rdfs:comment "This class represents the instant marking significant changes to a planning model before submission to review (BSTR = Before Submission To Review). Multiple events possible."@en ;

rdfs:label "Edit BSTR"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#GeologicalEvent

occp:GeologicalEvent rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseH\_Instant ;

rdfs:comment "This class represents the instant marking a geological event (e.g., earthquake) affecting a component or building. Multiple events possible."@en ;

rdfs:label "Geological Event"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#InspectionExecution

occp:InspectionExecution rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseG\_Instant ;

rdfs:comment "This class represents the instant marking the execution of a regular inspection. Multiple events possible over the lifecycle."@en ;

rdfs:label "Inspection Execution"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#IrregularInspection

occp:IrregularInspection rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseG\_Instant ;

rdfs:comment "This class represents the instant marking the execution of an irregular inspection. Multiple events possible over the lifecycle."@en ;

rdfs:label "Irregular Inspection"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#LifeCycle

occp:LifeCycle rdf:type owl:Class ;

rdfs:subClassOf time:Interval ;

rdfs:comment "This class represents a complete lifecycle (Phases A to M) within an overarching phase or another lifecycle, identified by a unique ID."@en ;

rdfs:label "Life Cycle"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#LoadEvent

occp:LoadEvent rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseH\_Instant ;

rdfs:comment "This class represents the instant marking an irregular load event affecting a component or building. Multiple events possible."@en ;

rdfs:label "Load Event"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#NewComponent

occp:NewComponent rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseL\_Instant ;

rdfs:comment "This class represents the instant marking the addition of a new component to an existing structure, potentially initiating a sub-lifecycle. Multiple events possible."@en ;

rdfs:label "New Component"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#NextRegularInspection

occp:NextRegularInspection rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseG\_Instant ;

rdfs:comment "This class represents the instant marking the planned date of the next regular inspection, based on the inspection interval. Multiple events possible over the lifecycle."@en ;

rdfs:label "Next Regular Inspection"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#Phase

occp:Phase rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Interval> ;

rdfs:comment "This class represents the main chronological blocks of a building’s lifecycle (e.g., planning, construction, usage), serving as a customizable base structure."@en ;

rdfs:label "Phase"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseA\_Instant

occp:PhaseA\_Instant rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This class represents instants within Phase A (Planning Phase), grouping all related events."@en ;

rdfs:label "Phase A Instant"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseA\_Planning

occp:PhaseA\_Planning rdf:type owl:Class ;

rdfs:subClassOf occp:Phase ;

rdfs:comment "This class represents the planning phase, starting with BeginningOfPlanning and ending with CompletionOfPlanning."@en ;

rdfs:label "Phase A Planning"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseB\_Instant

occp:PhaseB\_Instant rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This class represents instants within Phase B (Review Phase), grouping all related events."@en ;

rdfs:label "Phase B Instant"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseB\_Review

occp:PhaseB\_Review rdf:type owl:Class ;

rdfs:subClassOf occp:Phase ;

rdfs:comment "This class represents the review phase, starting with the first submission to review and ending with CompletionOfPlanning."@en ;

rdfs:label "Phase B Review"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseC\_Construction

occp:PhaseC\_Construction rdf:type owl:Class ;

rdfs:subClassOf occp:Phase ;

rdfs:comment "This class represents the construction phase, starting with ConstructionStart and ending with CompletionOfConstruction."@en ;

rdfs:label "Phase C Construction"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseC\_Instant

occp:PhaseC\_Instant rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This class represents instants within Phase C (Construction Phase), grouping all related events."@en ;

rdfs:label "Phase C Instant"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseD\_Instant

occp:PhaseD\_Instant rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This class represents instants within Phase D (Usage Phase), grouping all related events."@en ;

rdfs:label "Phase D Instant"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseD\_Usage

occp:PhaseD\_Usage rdf:type owl:Class ;

rdfs:subClassOf occp:Phase ;

rdfs:comment "This class represents the usage phase, starting with UsageStart and ending with UsageEnd."@en ;

rdfs:label "Phase D Usage"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseE\_Instant

occp:PhaseE\_Instant rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This class represents instants within Phase E (Warranty Phase), grouping all related events."@en ;

rdfs:label "Phase E Instant"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseE\_Warranty

occp:PhaseE\_Warranty rdf:type owl:Class ;

rdfs:subClassOf occp:Phase ;

rdfs:comment "This class represents the warranty phase, starting with WarrantyStart and ending with WarrantyEnd."@en ;

rdfs:label "Phase E Warranty"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseF\_DesignLife

occp:PhaseF\_DesignLife rdf:type owl:Class ;

rdfs:subClassOf occp:Phase ;

rdfs:comment "This class represents the design life phase, starting with DesignLifeStart and ending with DesignLifeEnd."@en ;

rdfs:label "Phase F Design Life"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseF\_Instant

occp:PhaseF\_Instant rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This class represents instants within Phase F (Design Life Phase), grouping all related events."@en ;

rdfs:label "Phase F Instant"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseG\_Inspection

occp:PhaseG\_Inspection rdf:type owl:Class ;

rdfs:subClassOf occp:Phase ;

rdfs:comment "This class represents the inspection phase, starting with the beginning of usage and ending with the beginning of deconstruction."@en ;

rdfs:label "Phase G Inspection"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseG\_Instant

occp:PhaseG\_Instant rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This class represents instants within Phase G (Inspection Phase), grouping all related events."@en ;

rdfs:label "Phase G Instant"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseH\_Instant

occp:PhaseH\_Instant rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This class represents instants within Phase H (Special Events Phase), grouping all related events."@en ;

rdfs:label "Phase H Instant"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseH\_SpecialEvents

occp:PhaseH\_SpecialEvents rdf:type owl:Class ;

rdfs:subClassOf occp:Phase ;

rdfs:comment "This class represents the special events phase, starting with the first special event and ending with the last."@en ;

rdfs:label "Phase H Special Events"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseK\_Damage

occp:PhaseK\_Damage rdf:type owl:Class ;

rdfs:subClassOf occp:Phase ;

rdfs:comment "This class represents the damage phase, starting with DamageDetection and ending with repair completion or deconstruction start."@en ;

rdfs:label "Phase K Damage"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseK\_Instant

occp:PhaseK\_Instant rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This class represents instants within Phase K (Damage Phase), grouping all related events."@en ;

rdfs:label "Phase K Instant"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseL\_Instant

occp:PhaseL\_Instant rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This class represents instants within Phase L (Repair Phase), grouping all related events."@en ;

rdfs:label "Phase L Instant"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseL\_Repair

occp:PhaseL\_Repair rdf:type owl:Class ;

rdfs:subClassOf occp:Phase ;

rdfs:comment "This class represents the repair phase, starting with RepairStart and ending with RepairCompletion."@en ;

rdfs:label "Phase L Repair"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseM\_Deconstruction

occp:PhaseM\_Deconstruction rdf:type owl:Class ;

rdfs:subClassOf occp:Phase ;

rdfs:comment "This class represents the deconstruction phase, starting with DeconstructionStart and ending with DeconstructionCompletion."@en ;

rdfs:label "Phase M Deconstruction"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#PhaseM\_Instant

occp:PhaseM\_Instant rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This class represents instants within Phase M (Deconstruction Phase), grouping all related events."@en ;

rdfs:label "Phase M Instant"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#RepairCompletion

occp:RepairCompletion rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseL\_Instant ;

rdfs:comment "This class represents the instant marking the completion of damage repair works for a component or building. Multiple events possible."@en ;

rdfs:label "Repair Completion"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#RepairStart

occp:RepairStart rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseL\_Instant ;

rdfs:comment "This class represents the instant marking the beginning of damage repair works for a component or building. Multiple events possible."@en ;

rdfs:label "Repair Start"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#ReviewApproval

occp:ReviewApproval rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseB\_Instant ;

rdfs:comment "This class represents the instant marking the reviewer’s acceptance of a planning model, ending the planning and review phases."@en ;

rdfs:label "Review Approval"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#ReviewRejection

occp:ReviewRejection rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseB\_Instant ;

rdfs:comment "This class represents the instant marking the reviewer’s rejection of a planning model. Multiple events possible."@en ;

rdfs:label "Review Rejection"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#ReviewStart

occp:ReviewStart rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseB\_Instant ;

rdfs:comment "This class represents the instant marking the start of the review process. Multiple events possible with rejections."@en ;

rdfs:label "Review Start"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#Submission

occp:Submission rdf:type owl:Class ;

rdfs:subClassOf occp:Transition ;

rdfs:comment "This class represents the instant marking the end of the tendering process, serving as a transition to the construction phase."@en ;

rdfs:label "Submission"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#SubmissionToReview

occp:SubmissionToReview rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseA\_Instant ;

rdfs:comment "This class represents the instant marking the submission of a planning draft to the reviewer, starting Phase B. Multiple events possible."@en ;

rdfs:label "Submission to Review"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#Transition

occp:Transition rdf:type owl:Class ;

rdfs:subClassOf <http://www.w3.org/2006/time#Instant> ;

rdfs:comment "This class represents key milestones or triggering events between phases, typically enabling the start of the next phase or marking phase completion."@en ;

rdfs:label "Transition"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#UsageChange

occp:UsageChange rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseD\_Instant ;

rdfs:comment "This class represents the instant marking a change in usage of a component or building, potentially ending one or more usage phases and initiating new ones. Multiple events possible."@en ;

rdfs:label "Usage Change"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#UsageEnd

occp:UsageEnd rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseD\_Instant ;

rdfs:comment "This class represents the instant marking the end of usage for a component or building. Multiple events possible."@en ;

rdfs:label "Usage End"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#UsageStart

occp:UsageStart rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseD\_Instant ;

rdfs:comment "This class represents the instant marking the beginning of usage for a component or building. Multiple events possible."@en ;

rdfs:label "Usage Start"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#WarrantyChange

occp:WarrantyChange rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseE\_Instant ;

rdfs:comment "This class represents the instant marking a change in warranty due to repairs or extensions. Multiple events possible."@en ;

rdfs:label "Warranty Change"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#WarrantyEnd

occp:WarrantyEnd rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseE\_Instant ;

rdfs:comment "This class represents the instant marking the end of a warranty period for a component or building. Multiple events possible due to restarts."@en ;

rdfs:label "Warranty End"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#WarrantyStart

occp:WarrantyStart rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseE\_Instant ;

rdfs:comment "This class represents the instant marking the beginning of a warranty period for a component or building. Multiple events possible due to restarts."@en ;

rdfs:label "Warranty Start"@en .

### http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#WeatherEvent

occp:WeatherEvent rdf:type owl:Class ;

rdfs:subClassOf occp:PhaseH\_Instant ;

rdfs:comment "This class represents the instant marking a weather event (e.g., flood) affecting a component or building. Multiple events possible."@en ;

rdfs:label "Weather Event"@en .

Alles klar, ich habe die TBox deiner OCCP (OCCP\_V0.3.ttl) aufgenommen und werde sie mir genau anschauen. Das ist ein super Ausgangspunkt für die Validierung! Ich sehe, dass du die Ontologie auf der W3C TIME-Ontologie aufbaust und eine Menge spezifischer Klassen und Properties für die chronologischen Phasen und Ereignisse im Bau-Lifecycle definiert hast – das passt perfekt zu deinem cMod-Konzept.

Bevor wir mit der Validierung loslegen, warte ich noch auf den restlichen Code (z. B. ABox-Daten oder dein Validierungsmodul), damit ich alles im Kontext betrachten kann. Ich vermute, du möchtest die Ontologie mit SHACL-Regeln oder einem Reasoner wie Pellet validieren, richtig? Wenn ja, können wir das Schritt für Schritt angehen – z. B. erst die Konsistenz der TBox prüfen und dann schauen, wie sie mit Instanzdaten (ABox) harmoniert.

Falls du schon konkrete Fragen oder Ziele für die Validierung hast (z. B. bestimmte Regeln, die geprüft werden sollen), lass es mich wissen! Ich bin bereit, dir dabei zu helfen, das Ganze sauber und effizient umzusetzen. Was kommt als Nächstes?

Hier die gebündelten SHACL-Shapes (OCCP\_SHACL.ttl):

@prefix

sh: [http://www.w3.org/ns/shacl#](http://www.w3.org/ns/shacl) .

@prefix

xsd: [http://www.w3.org/2001/XMLSchema#](http://www.w3.org/2001/XMLSchema) .

@prefix

occp: [http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#](http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod_V0.1) .

@prefix

time: [http://www.w3.org/2006/time#](http://www.w3.org/2006/time) .

@prefix

osh: [http://www.occpshape.de/shape#](http://www.occpshape.de/shape) .#################################################################SHACL Rules#################################################################GENERAL SHACL RULES

osh:GeneralDatePropertyShape a sh:PropertyShape ;

sh:path ( occp:hasActualTime occp:hasEstimatedTime ) ;

sh:datatype xsd:date ;

sh:message "Dates must be in YYYY-MM-DD format."@en ;

sh:severity sh:Violation .

osh:GeneralTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:property [

sh:path ( time:hasBeginning time:hasEnd ) ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:message "At least one 'time:hasBeginning' or 'time:hasEnd' must reference a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:message "Each entity must have at most one actual beginning via occp:hasActualBeginning."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:message "Each entity must have at most one actual end via occp:hasActualEnd."@en ;

sh:severity sh:Violation ;

] .

osh:InstantTimeShape a sh:NodeShape ;

sh:targetClass time:Instant ;

sh:property [

sh:path occp:hasActualTime ;

sh:datatype xsd:date ;

sh:maxCount 1 ;

sh:message "Each instant must have at most one actual timestamp via occp:hasActualTime of type xsd:date."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasEstimatedTime ;

sh:datatype xsd:date ;

sh:message "Each instant must have at most one estimated timestamp via occp:hasEstimatedTime of type xsd:date."@en ;

sh:severity sh:Violation ;

] .

CYCLE RELATED SHACL RULES

osh:CycleShape a sh:NodeShape ;

sh:targetClass occp:Cycle ;

sh:property [

sh:path occp:hasCycleNumber ;

sh:datatype xsd:integer ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:minInclusive 1 ;

sh:message "Each 'occp:Cycle' must specify exactly one positive integer via 'occp:hasCycleNumber' to indicate the number of iterations."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path time:hasBeginning ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:Cycle' must have exactly one 'time:hasBeginning' referencing a 'time:Instant' to mark its start."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path time:hasEnd ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:Cycle' must have exactly one 'time:hasEnd' referencing a 'time:Instant' to mark its end."@en ;

sh:severity sh:Violation ;

] .

osh:CycleAShape a sh:NodeShape ;

sh:targetClass occp:CycleA\_PlanningReview ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:or (

[ sh:class occp:BeginningOfPlanning ; ]

[ sh:class occp:SubmissionToReview ; ]

) ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:CycleA\_PlanningReview' must have exactly one actual beginning via 'occp:hasActualBeginning' referencing 'occp:SubmissionToReview' or 'occp:BeginningOfPlanning'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:or (

[ sh:class occp:ReviewApproval ; ]

[ sh:class occp:ReviewRejection ; ]

) ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:CycleA\_PlanningReview' must have exactly one 'occp:hasActualEnd' referencing an instant like 'occp:ReviewApproval' or 'occp:ReviewRejection'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:isInPhase ;

sh:or (

[ sh:class occp:PhaseA\_Planning ]

[ sh:class occp:PhaseB\_Review ]

) ;

sh:minCount 1 ;

sh:message "Each 'occp:CycleA\_PlanningReview' must be embedded in at least one 'occp:PhaseA\_Planning' via 'occp:isInPhase'."@en ;

sh:severity sh:Violation ;

] .

osh:LifeCycleShape a sh:NodeShape ;

sh:targetClass occp:LifeCycle ;

sh:property [

sh:path occp:hasLifeCycleID ;

sh:datatype xsd:string ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:LifeCycle' must have exactly one 'occp:hasLifeCycleID' of type xsd:string to provide a unique identifier."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:isInPhase ;

sh:class occp:Phase ;

sh:minCount 1 ;

sh:message "Each 'occp:LifeCycle' must be embedded in at least one 'occp:Phase' via 'occp:isInPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path time:hasBeginning ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:LifeCycle' must have exactly one 'time:hasBeginning' referencing a 'time:Instant' to mark its start."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path time:hasEnd ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:LifeCycle' must have exactly one 'time:hasEnd' referencing a 'time:Instant' to mark its end."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path time:hasBeginning ;

sh:sparql [

sh:select """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

PREFIX time: <http://www.w3.org/2006/time#>

SELECT $this WHERE {

$this time:hasBeginning ?start .

$this occp:isInPhase ?phase .

?phase time:hasEnd ?phaseEnd .

FILTER (?start > ?phaseEnd)

}

""" ;

sh:message "Each 'occp:LifeCycle' must have its 'time:hasBeginning' before or equal to the 'time:hasEnd' of the 'occp:Phase' it is embedded in via 'occp:isInPhase'."@en ;

] ;

sh:severity sh:Violation ;

] .

PHASE RELATED SHACL RULES

osh:ShapeOfPhaseA a sh:NodeShape ;

sh:targetClass occp:PhaseA\_Planning ;

sh:property [

sh:path occp:endsBefore ;

sh:class occp:PhaseC\_Construction ;

sh:message "Each 'occp:PhaseA\_Planning' must end before 'occp:PhaseC\_Construction' begins via 'occp:endsBefore'."@en ;

sh:severity sh:Warning

] ;

sh:property [

sh:path occp:endsBefore ;

sh:class occp:CompletionOfConstruction ;

sh:message "Each 'occp:PhaseA\_Planning' must end before 'occp:CompletionOfConstruction' via 'occp:endsBefore'."@en ;

sh:severity sh:Violation

] ;

sh:and (

[ sh:path occp:beginsBefore ; sh:class occp:PhaseB\_Review ]

[ sh:path occp:beginsBefore ; sh:class occp:PhaseC\_Construction ;

sh:message "Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseB\_Review' and 'occp:PhaseC\_Construction' via 'occp:beginsBefore'."@en ;

sh:severity sh:Violation ]

) ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:BeginningOfPlanning ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseA\_Planning' must have exactly one actual beginning via 'occp:hasActualBeginning' referencing 'occp:BeginningOfPlanning'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:or (

[ sh:class occp:ReviewApproval ]

[ sh:class occp:CompletionOfPlanning ]

) ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseA\_Planning' must have exactly one actual end via 'occp:hasActualEnd' referencing 'occp:ReviewApproval' or 'occp:CompletionOfPlanning'."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfPhaseB a sh:NodeShape ;

sh:targetClass occp:PhaseB\_Review ;

sh:or (

[ sh:path occp:before ; sh:class occp:PhaseC\_Construction ]

[ sh:path occp:beginsBefore ; sh:class occp:PhaseC\_Construction ;

sh:message "Each 'occp:PhaseB\_Review' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore' or occur entirely before it via 'occp:before'."@en ;

sh:severity sh:Violation ]

) ;

sh:or (

[ sh:path occp:before ; sh:class occp:CompletionOfPlanning ]

[ sh:path occp:endsWith ; sh:class occp:CompletionOfPlanning ;

sh:message "Each 'occp:PhaseB\_Review' must end with 'occp:CompletionOfPlanning' via 'occp:endsWith' or occur entirely before it via 'occp:before'."@en ;

sh:severity sh:Violation ]

) ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:SubmissionToReview ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseB\_Review' can have only one actual beginning via 'occp:hasActualBeginning' referencing 'occp:SubmissionToReview'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:or (

[ sh:class occp:ReviewApproval ]

[ sh:class occp:CompletionOfPlanning ]

) ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseB\_Review' can have only one actual end via 'occp:hasActualEnd' referencing 'occp:ReviewApproval' or 'occp:CompletionOfPlanning'."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfTransitionA a sh:NodeShape ; # Transition A (Completion of Planning)

sh:targetClass occp:CompletionOfPlanning ;

sh:property [ # Transition order

sh:path occp:before ;

sh:class occp:BeginOfTenderingProcess ;

sh:message "Each 'occp:CompletionOfPlanning' should occur before 'occp:BeginOfTenderingProcess' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [ # Transition time (single point in time)

sh:path occp:hasActualTime ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:CompletionOfPlanning' must have exactly one actual timestamp via 'occp:hasActualTime' of type xsd:date."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfTransitionB a sh:NodeShape ; # Transition B (Begin Of Tendering Process)

sh:targetClass occp:BeginOfTenderingProcess ;

sh:property [ # Transition order

sh:path occp:before ;

sh:class occp:Submission ;

sh:message "Each 'occp:BeginOfTenderingProcess' should occur before 'occp:Submission' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [ # Transition time (single point in time)

sh:path occp:hasActualTime ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:BeginOfTenderingProcess' must have exactly one actual timestamp via 'occp:hasActualTime' of type xsd:date."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfTransitionC a sh:NodeShape ; # Transition C (Submission)

sh:targetClass occp:Submission ;

sh:property [ # Transition order

sh:path occp:before ;

sh:class occp:PhaseC\_Construction ;

sh:message "Each 'occp:Submission' should occur before 'occp:PhaseC\_Construction' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [ # Transition time (single point in time)

sh:path occp:hasActualTime ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each 'occp:Submission' must have exactly one actual timestamp via 'occp:hasActualTime' of type xsd:date."@en ;

] .

osh:ShapeOfPhaseC a sh:NodeShape ; # Phase C (Construction)

sh:targetClass occp:PhaseC\_Construction ;

sh:property [ # Phase order

sh:path occp:after ;

sh:class occp:CompletionOfPlanning ;

sh:message "Each 'occp:PhaseC\_Construction' should occur after 'occp:CompletionOfPlanning' via 'occp:after'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [ # Phase order

sh:path occp:after ;

sh:class occp:Submission ;

sh:message "Each 'occp:PhaseC\_Construction' should occur after 'occp:Submission' via 'occp:after'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:ConstructionStart ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseC\_Construction' can have only one actual beginning via 'occp:hasActualBeginning' referencing 'occp:ConstructionStart'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class occp:CompletionOfConstruction ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseC\_Construction' can have only one actual end via 'occp:hasActualEnd' referencing 'occp:CompletionOfConstruction'."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfTransitionD a sh:NodeShape ; # Transition D (Completion Of Construction)

sh:targetClass occp:CompletionOfConstruction ;

sh:property [ # Phase order

sh:path occp:after ;

sh:class occp:CompletionOfPlanning ;

sh:message "Each 'occp:CompletionOfConstruction' should occur after 'occp:CompletionOfPlanning' via 'occp:after'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [ # Phase order

sh:path occp:before ;

sh:class occp:PhaseD\_Usage ;

sh:message "Each 'occp:CompletionOfConstruction' should occur before 'occp:PhaseD\_Usage' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [ # Phase order

sh:path occp:before ;

sh:class occp:PhaseM\_Deconstruction ;

sh:message "Each 'occp:CompletionOfConstruction' should occur before 'occp:PhaseM\_Deconstruction' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [

sh:path occp:hasActualTime ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:CompletionOfConstruction' must have exactly one actual timestamp via 'occp:hasActualTime' of type xsd:date."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfPhaseD a sh:NodeShape ; # Phase D (Usage)

sh:targetClass occp:PhaseD\_Usage ;

sh:property [ # Phase order

sh:path occp:after ;

sh:class occp:CompletionOfConstruction ;

sh:message "Each 'occp:PhaseD\_Usage' should occur after 'occp:CompletionOfConstruction' via 'occp:after'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [ # Phase order

sh:path occp:endsBefore ;

sh:class occp:PhaseM\_Deconstruction ;

sh:message "Each 'occp:PhaseD\_Usage' should end before 'occp:PhaseM\_Deconstruction' via 'occp:endsBefore'. We don’t want anyone to get hurt, do we?"@en ;

sh:severity sh:Warning ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:UsageStart ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseD\_Usage' can have only one actual beginning via 'occp:hasActualBeginning' referencing 'occp:UsageStart'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class occp:UsageEnd ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseD\_Usage' can have only one actual end via 'occp:hasActualEnd' referencing 'occp:UsageEnd'."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfPhaseE a sh:NodeShape ; # Phase E (Warranty)

sh:targetClass occp:PhaseE\_Warranty ;

sh:property [ # Phase order

sh:path occp:after ;

sh:class occp:CompletionOfConstruction ;

sh:message "Each 'occp:PhaseE\_Warranty' should occur after 'occp:CompletionOfConstruction' via 'occp:after'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [ # Begin date

sh:path occp:hasActualBeginning ;

sh:class occp:WarrantyStart ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseE\_Warranty' can have only one actual beginning via 'occp:hasActualBeginning' referencing 'occp:WarrantyStart'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # End date

sh:path occp:hasActualEnd ;

sh:class occp:WarrantyEnd ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseE\_Warranty' can have only one actual end via 'occp:hasActualEnd' referencing 'occp:WarrantyEnd'."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfPhaseF a sh:NodeShape ; # Phase F (DesignLife)

sh:targetClass occp:PhaseF\_DesignLife ;

sh:property [ # Phase order

sh:path occp:after ;

sh:class occp:CompletionOfConstruction ;

sh:message "Each 'occp:PhaseF\_DesignLife' should occur after 'occp:CompletionOfConstruction' via 'occp:after'."@en ;

sh:severity sh:Warning ;

] ;

sh:or ( # Phase order options - Phase F

[

sh:path occp:beginsBefore ;

sh:class occp:PhaseD\_Usage ;

]

[

sh:path occp:beginsWith ;

sh:class occp:PhaseD\_Usage ;

sh:message "Each 'occp:PhaseF\_DesignLife' should begin before 'occp:PhaseD\_Usage' via 'occp:beginsBefore' or with 'occp:PhaseD\_Usage' via 'occp:beginsWith'."@en ;

sh:severity sh:Warning ;

]

) ;

sh:property [

sh:path occp:before ;

sh:class occp:PhaseM\_Deconstruction ;

sh:message "Each 'occp:PhaseF\_DesignLife' should occur before 'occp:PhaseM\_Deconstruction' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:DesignLifeStart ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseF\_DesignLife' can only have one actual beginning via 'occp:hasActualBeginning' referencing 'occp:DesignLifeStart'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class occp:DesignLifeEnd ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseF\_DesignLife' can only have one actual end via 'occp:hasActualEnd' referencing 'occp:DesignLifeEnd'."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfPhaseG a sh:NodeShape ; # Phase G (Inspection)

sh:targetClass occp:PhaseG\_Inspection ;

sh:property [

sh:or (

[ sh:path occp:beginsWith ; sh:class occp:PhaseD\_Usage ]

[ sh:path occp:beginsAfter ; sh:class occp:UsageStart ]

) ;

sh:message "Each 'occp:PhaseG\_Inspection' must begin with 'occp:PhaseD\_Usage' via 'occp:beginsWith' or after 'occp:UsageStart' via 'occp:beginsAfter'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:after ;

sh:class occp:CompletionOfConstruction ;

sh:message "Each 'occp:PhaseG\_Inspection' should occur after 'occp:CompletionOfConstruction' via 'occp:after'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:PhaseM\_Deconstruction ;

sh:message "Each 'occp:PhaseG\_Inspection' should occur before 'occp:PhaseM\_Deconstruction' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:InspectionExecution ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseG\_Inspection' can only have one actual beginning via 'occp:hasActualBeginning' referencing 'occp:InspectionExecution'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseG\_Inspection' can only have one actual end via 'occp:hasActualEnd' referencing a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfPhaseH a sh:NodeShape ; # Phase H (SpecialEvents) - Special events can occur anytime, so this is to be handled like a super-phase. It can be used to describe or record events that have shaped the landscape like earthquakes in the past, but also events that occure during or after the construction's life cycle.

sh:targetClass occp:PhaseH\_SpecialEvents ;

sh:property [

sh:path occp:hasInstant ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:message "Each 'occp:PhaseH\_SpecialEvents' must have at least one instant via 'occp:hasInstant' referencing a 'time:Instant' (e.g., 'occp:GeologicalEvent', 'occp:WeatherEvent')."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseH\_SpecialEvents' can only have one actual beginning via 'occp:hasActualBeginning' referencing a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseH\_SpecialEvents' can only have one actual end via 'occp:hasActualEnd' referencing a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfPhaseK a sh:NodeShape ; # Phase K (Damage)

sh:targetClass occp:PhaseK\_Damage ;

sh:property [ # Phase order

sh:path occp:after ;

sh:class occp:Submission ;

sh:message "Each 'occp:PhaseK\_Damage' should occur after 'occp:Submission' via 'occp:after'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [ # Phase order

sh:path occp:before ;

sh:class occp:CompletionOfDeconstruction ;

sh:message "Each 'occp:PhaseK\_Damage' should occur before 'occp:CompletionOfDeconstruction' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [

sh:path occp:beginsBefore ;

sh:class occp:PhaseL\_Repair ;

sh:message "Each 'occp:PhaseK\_Damage' should begin before 'occp:PhaseL\_Repair' via 'occp:beginsBefore', but damage detection during repair is possible."@en ;

sh:severity sh:Warning ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:DamageDetection ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseK\_Damage' can only have one actual beginning via 'occp:hasActualBeginning' referencing 'occp:DamageDetection'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseK\_Damage' can only have one actual end via 'occp:hasActualEnd' referencing a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfPhaseL a sh:NodeShape ; # Phase L (Repair)

sh:targetClass occp:PhaseL\_Repair ;

sh:property [

sh:or (

[ sh:path occp:endsAfter ; sh:class occp:PhaseK\_Damage ]

[ sh:path occp:endsWith ; sh:class occp:PhaseK\_Damage ]

) ;

sh:message "Each 'occp:PhaseL\_Repair' must end after 'occp:PhaseK\_Damage' via 'occp:endsAfter' or with 'occp:PhaseK\_Damage' via 'occp:endsWith' to ensure repair resolves damage."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:after ;

sh:class occp:Submission ;

sh:message "Each 'occp:PhaseL\_Repair' should occur after 'occp:Submission' via 'occp:after'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:PhaseM\_Deconstruction ;

sh:message "Each 'occp:PhaseL\_Repair' should occur before 'occp:PhaseM\_Deconstruction' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:RepairStart ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseL\_Repair' can only have one actual beginning via 'occp:hasActualBeginning' referencing 'occp:RepairStart'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class occp:RepairCompletion ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseL\_Repair' can only have one actual end via 'occp:hasActualEnd' referencing 'occp:RepairCompletion'."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfPhaseM a sh:NodeShape ; # Phase M (Deconstruction)

sh:targetClass occp:PhaseM\_Deconstruction ;

sh:property [ # Phase order

sh:path occp:after ;

sh:class occp:CompletionOfConstruction ;

sh:message "Each 'occp:PhaseM\_Deconstruction' should occur after 'occp:CompletionOfConstruction' via 'occp:after'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:DeconstructionStart ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseM\_Deconstruction' can only have one actual beginning via 'occp:hasActualBeginning' referencing 'occp:DeconstructionStart'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class occp:CompletionOfDeconstruction ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseM\_Deconstruction' can only have one actual end via 'occp:hasActualEnd' referencing 'occp:CompletionOfDeconstruction'."@en ;

sh:severity sh:Violation ;

] .

osh:ShapeOfTransitionE a sh:NodeShape ; # Transition E (Completion Of Deconstruction)

sh:targetClass occp:CompletionOfDeconstruction ;

sh:or ( # Transition order options

[

sh:path occp:after ;

sh:class occp:PhaseM\_Deconstruction ;

]

[

sh:path occp:endsWith ;

sh:class occp:PhaseM\_Deconstruction ;

sh:message "Each 'occp:CompletionOfDeconstruction' must occur after 'occp:PhaseM\_Deconstruction' via 'occp:after' or end with 'occp:PhaseM\_Deconstruction' via 'occp:endsWith'."@en ;

sh:severity sh:Violation ;

]

) ;

sh:property [ # Transition time (single point in time)

sh:path occp:hasActualTime ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:CompletionOfDeconstruction' must have exactly one actual timestamp via 'occp:hasActualTime' of type xsd:date."@en ;

sh:severity sh:Violation ;

] .

INSTANT RELATED SHACL RULES

### PHASE A INSTANTS

# PHASE ASSIGNMENT

osh:PhaseAInstantShape a sh:NodeShape ;

sh:targetClass occp:PhaseA\_Instant ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseA\_Planning ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseA\_Instant' must belong to exactly one 'occp:PhaseA\_Planning' via 'occp:hasPhase'."@en ;

sh:severity sh:Violation ;

] .

### INSTANT SHAPES PHASE A - PLANNING

osh:BeginningOfPlanningShape a sh:NodeShape ; # Instant Beginning Of Planning (Phase A - Planning)

sh:targetClass occp:BeginningOfPlanning ;

sh:property [ # Instant (Beginning Of Planning) starts a new Phase A (Planning).

sh:path occp:startsPhase ;

sh:class occp:PhaseA\_Planning ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:BeginningOfPlanning' must start exactly one 'occp:PhaseA\_Planning' via 'occp:startsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:before ;

sh:class occp:DataProcurement ;

sh:message "Each 'occp:BeginningOfPlanning' must occur before 'occp:DataProcurement' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:SubmissionToReview ;

sh:message "Each 'occp:BeginningOfPlanning' should occur before 'occp:SubmissionToReview' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] .

osh:DataProcurementShape a sh:NodeShape ; # Instant Data Procurement (Phase A - Planning)

sh:targetClass occp:DataProcurement ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:after ;

sh:class occp:BeginningOfPlanning ;

sh:message "Each 'occp:DataProcurement' must occur after 'occp:BeginningOfPlanning' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:before ;

sh:class occp:SubmissionToReview ;

sh:message "Each 'occp:DataProcurement' should occur before 'occp:SubmissionToReview' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] .

osh:Edit\_BSTRShape a sh:NodeShape ; # Instant Edit Before Submission To Review (Phase A - Planning)

sh:targetClass occp:Edit\_BSTR ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:after ;

sh:class occp:BeginningOfPlanning ;

sh:message "Each 'occp:Edit\_BSTR' must occur after 'occp:BeginningOfPlanning' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:before ;

sh:class occp:SubmissionToReview ;

sh:message "Each 'occp:Edit\_BSTR' must occur before 'occp:SubmissionToReview' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:SubmissionToReviewShape a sh:NodeShape ; # Instant Submission To Review (Phase A - Planning)

sh:targetClass occp:SubmissionToReview ;

sh:property [ # Instant's chronological order within the Phase - 2 Options

sh:path occp:after ;

sh:or (

[ sh:class occp:Edit\_BSTR ]

[ sh:class occp:Edit\_ASTR ]

) ;

sh:message "Each 'occp:SubmissionToReview' must occur after 'occp:Edit\_BSTR' or 'occp:Edit\_ASTR' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:startsCycle ;

sh:class occp:CycleA\_PlanningReview ;

sh:maxCount 1 ;

sh:message "Each 'occp:SubmissionToReview' can start only one 'occp:CycleA\_PlanningReview' via 'occp:startsCycle'."@en ;

sh:severity sh:Violation ;

] .

osh:Edit\_ASTRShape a sh:NodeShape ; # Instant Edit After Submission To Review (Phase A - Planning)

sh:targetClass occp:Edit\_ASTR ;

sh:property [

sh:path occp:after ;

sh:class occp:SubmissionToReview ;

sh:message "Each 'occp:Edit\_ASTR' must occur after 'occp:SubmissionToReview' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:CompletionOfPlanning ;

sh:message "Each 'occp:Edit\_ASTR' should occur before 'occp:CompletionOfPlanning' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] .

### PHASE B INSTANTS

# PHASE ASSIGNMENT

osh:PhaseBInstantShape a sh:NodeShape ;

sh:targetClass occp:PhaseB\_Instant ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseB\_Review ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseB\_Instant' must belong to exactly one 'occp:PhaseB\_Review' via 'occp:hasPhase'."@en ;

sh:severity sh:Violation ;

] .

### INSTANT SHAPES PHASE B - REVIEW

osh:ReviewStartShape a sh:NodeShape ; # Instant Review Start (Phase B - Review)

sh:targetClass occp:ReviewStart ;

sh:property [

sh:path occp:startsPhase ;

sh:class occp:PhaseB\_Review ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:ReviewStart' must start exactly one 'occp:PhaseB\_Review' via 'occp:startsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:or ( # Instant's chronological order within the Phase - 2 Options

[

sh:path occp:before ;

sh:class occp:ReviewApproval ;

sh:message "Each 'occp:ReviewStart' must occur before 'occp:ReviewApproval' via 'occp:before'."@en ;

sh:severity sh:Violation ;

]

[

sh:path occp:before ;

sh:class occp:ReviewRejection ;

sh:message "Each 'occp:ReviewStart' must occur before 'occp:ReviewRejection' via 'occp:before'."@en ;

sh:severity sh:Violation ;

]

) ;

sh:property [

sh:path occp:after ;

sh:class occp:SubmissionToReview ;

sh:message "Each 'occp:ReviewStart' must occur after 'occp:SubmissionToReview' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] .

osh:ReviewApprovalShape a sh:NodeShape ; # Instant Review Approval (Phase B - Review)

sh:targetClass occp:ReviewApproval ;

sh:property [

sh:path occp:endsPhase ;

sh:class occp:PhaseA\_Planning ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:ReviewApproval' must end exactly one 'occp:PhaseA\_Planning' via 'occp:endsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:endsPhase ;

sh:class occp:PhaseB\_Review ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:ReviewApproval' must end exactly one 'occp:PhaseB\_Review' via 'occp:endsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:after ;

sh:class occp:ReviewStart ;

sh:message "Each 'occp:ReviewApproval' must occur after 'occp:ReviewStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant (Review Approval) ends current Cycle A (Planning/Review).

sh:path occp:endsCycle ;

sh:class occp:CycleA\_PlanningReview ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:ReviewApproval' must end exactly one 'occp:CycleA\_PlanningReview' via 'occp:endsCycle'."@en ;

sh:severity sh:Violation ;

] .

osh:ReviewRejectionShape a sh:NodeShape ; # Instant Review Rejection (Phase B - Review)

sh:targetClass occp:ReviewRejection ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:after ;

sh:class occp:ReviewStart ;

sh:message "Each 'occp:ReviewRejection' must occur after 'occp:ReviewStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:endsCycle ;

sh:class occp:CycleA\_PlanningReview ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:ReviewRejection' must end exactly one 'occp:CycleA\_PlanningReview' via 'occp:endsCycle'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:startsCycle ;

sh:class occp:CycleA\_PlanningReview ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:ReviewRejection' must start exactly one new 'occp:CycleA\_PlanningReview' via 'occp:startsCycle'."@en ;

sh:severity sh:Violation ;

] .

### PHASE C INSTANTS

# PHASE ASSIGNMENT

osh:PhaseCInstantShape a sh:NodeShape ;

sh:targetClass occp:PhaseC\_Instant ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseC\_Construction ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseC\_Instant' must belong to exactly one 'occp:PhaseC\_Construction' via 'occp:hasPhase'."@en ;

sh:severity sh:Violation ;

] .

### INSTANT SHAPES PHASE C - CONSTRUCTION

osh:ConstructionStartShape a sh:NodeShape ; # Instant Construction Start (Phase C - Construction)

sh:targetClass occp:ConstructionStart ;

sh:property [

sh:path occp:startsPhase ;

sh:class occp:PhaseC\_Construction ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:ConstructionStart' must start exactly one 'occp:PhaseC\_Construction' via 'occp:startsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:before ;

sh:class occp:ConstructionCompletion ;

sh:message "Each 'occp:ConstructionStart' must occur before 'occp:ConstructionCompletion' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:after ;

sh:class occp:Submission ;

sh:message "Each 'occp:ConstructionStart' must occur after 'occp:Submission' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] .

osh:ConstructionCompletionShape a sh:NodeShape ; # Instant Construction Completion (Phase C - Construction)

sh:targetClass occp:ConstructionCompletion ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:after ;

sh:class occp:ConstructionStart ;

sh:message "Each 'occp:ConstructionCompletion' must occur after 'occp:ConstructionStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:or ( # Instant's chronological order within the Phase - 2 Options

[

sh:path occp:before ;

sh:class occp:ConstructionAcceptance ;

sh:message "Each 'occp:ConstructionCompletion' must occur before 'occp:ConstructionAcceptance' via 'occp:before'."@en ;

sh:severity sh:Violation ;

]

[

sh:path occp:before ;

sh:class occp:DefectElimStart ;

sh:message "Each 'occp:ConstructionCompletion' must occur before 'occp:DefectElimStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

]

) .

osh:DefectElimStartShape a sh:NodeShape ; # Instant Defect Elimination Start (Phase C - Construction)

sh:targetClass occp:DefectElimStart ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:after ;

sh:class occp:ConstructionCompletion ;

sh:message "Each 'occp:DefectElimStart' must occur after 'occp:ConstructionCompletion' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DefectElimCompletion ;

sh:message "Each 'occp:DefectElimStart' must occur before 'occp:DefectElimCompletion' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:DefectElimCompletionShape a sh:NodeShape ; # Instant Defect Elimination Completion (Phase C - Construction)

sh:targetClass occp:DefectElimCompletion ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:after ;

sh:class occp:DefectElimStart ;

sh:message "Each 'occp:DefectElimCompletion' must occur after 'occp:DefectElimStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:ConstructionAcceptance ;

sh:message "Each 'occp:DefectElimCompletion' must occur before 'occp:ConstructionAcceptance' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:ConstructionAcceptanceShape a sh:NodeShape ; # Instant Construction Acceptance (Phase C - Construction)

sh:targetClass occp:ConstructionAcceptance ;

sh:property [ # Instant's chronological order

sh:path occp:before ;

sh:class occp:UsageStart ;

sh:message "Each 'occp:ConstructionAcceptance' should occur before 'occp:UsageStart' via 'occp:before'."@en ;

sh:severity sh:Warning ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:CompletionOfConstruction ;

sh:message "Each 'occp:ConstructionAcceptance' must occur before 'occp:CompletionOfConstruction' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] ;

sh:or ( # Instant's chronological order within the Phase - 2 Options

[

sh:path occp:after ;

sh:class occp:ConstructionCompletion ;

sh:message "Each 'occp:ConstructionAcceptance' must occur after 'occp:ConstructionCompletion' via 'occp:after'."@en ;

sh:severity sh:Violation ;

]

[

sh:path occp:after ;

sh:class occp:DefectElimCompletion ;

sh:message "Each 'occp:ConstructionAcceptance' must occur after 'occp:DefectElimCompletion' via 'occp:after'."@en ;

sh:severity sh:Violation ;

]

) .

### PHASE D INSTANTS

# PHASE ASSIGNMENT

osh:PhaseDInstantShape a sh:NodeShape ;

sh:targetClass occp:PhaseD\_Instant ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseD\_Usage ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseD\_Instant' must belong to exactly one 'occp:PhaseD\_Usage' via 'occp:hasPhase'."@en ;

sh:severity sh:Violation ;

] .

### INSTANT SHAPES PHASE D - USAGE

osh:UsageStartShape a sh:NodeShape ; # Instant Usage Start (Phase D - Usage)

sh:targetClass occp:UsageStart ;

sh:property [ # A new Usage Phase is started with Usage Start

sh:path occp:startsPhase ;

sh:class occp:PhaseD\_Usage ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:UsageStart' must start exactly one 'occp:PhaseD\_Usage' via 'occp:startsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:ConstructionAcceptance ;

sh:message "Each 'occp:UsageStart' must occur after 'occp:ConstructionAcceptance' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:UsageEnd ;

sh:message "Each 'occp:UsageStart' must occur before 'occp:UsageEnd' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:UsageChangeShape a sh:NodeShape ; # Instant Usage Change (Phase D - Usage)

sh:targetClass occp:UsageChange ;

sh:property [ # The current Usage Phase is ended with a Change of Usage

sh:path occp:endsPhase ;

sh:class occp:PhaseD\_Usage ;

sh:minCount 1 ;

sh:message "Each 'occp:UsageChange' must end at least one 'occp:PhaseD\_Usage' via 'occp:endsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # A new Usage Phase is started with a Change of Usage

sh:path occp:startsPhase ;

sh:class occp:PhaseD\_Usage ;

sh:minCount 1 ;

sh:message "Each 'occp:UsageChange' must start at least one 'occp:PhaseD\_Usage' via 'occp:startsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:CompletionOfConstruction ;

sh:message "Each 'occp:UsageChange' must occur after 'occp:CompletionOfConstruction' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:after ;

sh:class occp:UsageStart ;

sh:message "Each 'occp:UsageChange' must occur after 'occp:UsageStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:UsageEnd ;

sh:message "Each 'occp:UsageChange' must occur before 'occp:UsageEnd' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:UsageChange' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:UsageEndShape a sh:NodeShape ; # Instant Usage End (Phase D - Usage)

sh:targetClass occp:UsageEnd ;

sh:property [

sh:path occp:endsPhase ;

sh:class occp:PhaseD\_Usage ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:UsageEnd' must end exactly one 'occp:PhaseD\_Usage' via 'occp:endsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:or ( # Instant's chronological order within the Phase - 2 Options

[

sh:path occp:after ;

sh:class occp:UsageStart ;

sh:message "Each 'occp:UsageEnd' must occur after 'occp:UsageStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

]

[

sh:path occp:after ;

sh:class occp:UsageChange ;

sh:message "Each 'occp:UsageEnd' must occur after 'occp:UsageChange' via 'occp:after'."@en ;

sh:severity sh:Violation ;

]

) ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:UsageEnd' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

### PHASE E INSTANTS

# PHASE ASSIGNMENT

osh:PhaseEInstantShape a sh:NodeShape ;

sh:targetClass occp:PhaseE\_Instant ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseE\_Warranty ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseE\_Instant' must belong to exactly one 'occp:PhaseE\_Warranty' via 'occp:hasPhase'."@en ;

sh:severity sh:Violation ;

] .

### INSTANT SHAPES PHASE E - WARRANTY

osh:WarrantyStartShape a sh:NodeShape ; # Instant Warranty Start (Phase E - Warranty)

sh:targetClass occp:WarrantyStart ;

sh:property [ # A new Warranty Phase is started with Warranty Start

sh:path occp:startsPhase ;

sh:class occp:PhaseE\_Warranty ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:WarrantyStart' must start exactly one 'occp:PhaseE\_Warranty' via 'occp:startsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:ConstructionAcceptance ;

sh:message "Each 'occp:WarrantyStart' must occur after 'occp:ConstructionAcceptance' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:WarrantyEnd ;

sh:message "Each 'occp:WarrantyStart' must occur before 'occp:WarrantyEnd' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:WarrantyChangeShape a sh:NodeShape ; # Instant Warranty Change (Phase E - Warranty)

sh:targetClass occp:WarrantyChange ;

sh:property [ # The current Warranty Phase is ended with a Change of Warranty

sh:path occp:endsPhase ;

sh:class occp:PhaseE\_Warranty ;

sh:minCount 1 ;

sh:message "Each 'occp:WarrantyChange' must end at least one 'occp:PhaseE\_Warranty' via 'occp:endsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # A new Warranty Phase is started with a Change of Warranty

sh:path occp:startsPhase ;

sh:class occp:PhaseE\_Warranty ;

sh:minCount 1 ;

sh:message "Each 'occp:WarrantyChange' must start at least one 'occp:PhaseE\_Warranty' via 'occp:startsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:after ;

sh:class occp:CompletionOfConstruction ;

sh:message "Each 'occp:WarrantyChange' must occur after 'occp:CompletionOfConstruction' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:after ;

sh:class occp:WarrantyStart ;

sh:message "Each 'occp:WarrantyChange' must occur after 'occp:WarrantyStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:WarrantyEnd ;

sh:message "Each 'occp:WarrantyChange' must occur before 'occp:WarrantyEnd' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:WarrantyChange' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:WarrantyEndShape a sh:NodeShape ; # Instant Warranty End (Phase E - Warranty)

sh:targetClass occp:WarrantyEnd ;

sh:property [ # The current Warranty Phase is ended with Warranty End

sh:path occp:endsPhase ;

sh:class occp:PhaseE\_Warranty ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:WarrantyEnd' must end exactly one 'occp:PhaseE\_Warranty' via 'occp:endsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:or ( # Instant's chronological order within the Phase - 2 Options

[

sh:path occp:after ;

sh:class occp:WarrantyStart ;

sh:message "Each 'occp:WarrantyEnd' must occur after 'occp:WarrantyStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

]

[

sh:path occp:after ;

sh:class occp:WarrantyChange ;

sh:message "Each 'occp:WarrantyEnd' must occur after 'occp:WarrantyChange' via 'occp:after'."@en ;

sh:severity sh:Violation ;

]

) ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:WarrantyEnd' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

### PHASE F INSTANTS

# PHASE ASSIGNMENT

osh:PhaseFInstantShape a sh:NodeShape ;

sh:targetClass occp:PhaseF\_Instant ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseF\_DesignLife ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseF\_Instant' must belong to exactly one 'occp:PhaseF\_DesignLife' via 'occp:hasPhase'."@en ;

sh:severity sh:Violation ;

] .

### INSTANT SHAPES PHASE F - DESIGN LIFE

osh:DesignLifeStartShape a sh:NodeShape ; # Instant Design Life Start (Phase F - Design Life)

sh:targetClass occp:DesignLifeStart ;

sh:property [ # A new Design Life Phase is started with Design Life Start

sh:path occp:startsPhase ;

sh:class occp:PhaseF\_DesignLife ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:DesignLifeStart' must start exactly one 'occp:PhaseF\_DesignLife' via 'occp:startsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:ConstructionAcceptance ;

sh:message "Each 'occp:DesignLifeStart' must occur after 'occp:ConstructionAcceptance' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:DesignLifeStart' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DesignLifeEnd ;

sh:message "Each 'occp:DesignLifeStart' must occur before 'occp:DesignLifeEnd' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:DesignLifeChangeShape a sh:NodeShape ; # Instant Design Life Change (Phase F - Design Life)

sh:targetClass occp:DesignLifeChange ;

sh:property [ # The current Design Life Phase is ended with a Change of Design Life

sh:path occp:endsPhase ;

sh:class occp:PhaseF\_DesignLife ;

sh:minCount 1 ;

sh:message "Each 'occp:DesignLifeChange' must end at least one 'occp:PhaseF\_DesignLife' via 'occp:endsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # A new Design Life Phase is started with a Change of Design Life

sh:path occp:startsPhase ;

sh:class occp:PhaseF\_DesignLife ;

sh:minCount 1 ;

sh:message "Each 'occp:DesignLifeChange' must start at least one 'occp:PhaseF\_DesignLife' via 'occp:startsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:ConstructionAcceptance ;

sh:message "Each 'occp:DesignLifeChange' must occur after 'occp:ConstructionAcceptance' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order within the Phase

sh:path occp:after ;

sh:class occp:DesignLifeStart ;

sh:message "Each 'occp:DesignLifeChange' must occur after 'occp:DesignLifeStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DesignLifeEnd ;

sh:message "Each 'occp:DesignLifeChange' must occur before 'occp:DesignLifeEnd' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:DesignLifeChange' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:DesignLifeEndShape a sh:NodeShape ; # Instant Design Life End (Phase F - Design Life)

sh:targetClass occp:DesignLifeEnd ;

sh:property [ # The current Design Life Phase is ended with Design Life End

sh:path occp:endsPhase ;

sh:class occp:PhaseF\_DesignLife ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:DesignLifeEnd' must end exactly one 'occp:PhaseF\_DesignLife' via 'occp:endsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:or ( # Instant's chronological order within the Phase - 2 Options

[

sh:path occp:after ;

sh:class occp:DesignLifeStart ;

sh:message "Each 'occp:DesignLifeEnd' must occur after 'occp:DesignLifeStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

]

[

sh:path occp:after ;

sh:class occp:DesignLifeChange ;

sh:message "Each 'occp:DesignLifeEnd' must occur after 'occp:DesignLifeChange' via 'occp:after'."@en ;

sh:severity sh:Violation ;

]

) ;

sh:property [

sh:path occp:after ;

sh:class occp:ConstructionAcceptance ;

sh:message "Each 'occp:DesignLifeEnd' must occur after 'occp:ConstructionAcceptance' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:DesignLifeEnd' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

### PHASE G INSTANTS

# PHASE ASSIGNMENT

osh:PhaseGInstantShape a sh:NodeShape ;

sh:targetClass occp:PhaseG\_Instant ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseG\_Inspection ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseG\_Instant' must belong to exactly one 'occp:PhaseG\_Inspection' via 'occp:hasPhase'."@en ;

sh:severity sh:Violation ;

] .

### INSTANT SHAPES PHASE G - INSPECTION

osh:InspectionExecutionShape a sh:NodeShape ; # Instant Inspection Execution (Phase G - Inspection)

sh:targetClass occp:InspectionExecution ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:ConstructionStart ;

sh:message "Each 'occp:InspectionExecution' must occur after 'occp:ConstructionStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:or ( # Instant's chronological order - 2 Options

[

sh:path occp:after ;

sh:class occp:UsageStart ;

sh:message "Each 'occp:InspectionExecution' must occur after 'occp:UsageStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

]

[

sh:path occp:after ;

sh:class occp:UsageChange ;

sh:message "Each 'occp:InspectionExecution' must occur after 'occp:UsageChange' via 'occp:after'."@en ;

sh:severity sh:Violation ;

]

) ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:InspectionExecution' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:IrregularInspectionShape a sh:NodeShape ; # Instant Irregular Inspection (Phase G - Inspection)

sh:targetClass occp:IrregularInspection ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:UsageStart ;

sh:message "Each 'occp:IrregularInspection' must occur after 'occp:UsageStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:IrregularInspection' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:NextRegularInspectionShape a sh:NodeShape ; # Instant Next Regular Inspection (Phase G - Inspection)

sh:targetClass occp:NextRegularInspection ;

sh:property [

sh:path occp:after ;

sh:class occp:InspectionExecution ;

sh:message "Each 'occp:NextRegularInspection' must occur after 'occp:InspectionExecution' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:UsageStart ;

sh:message "Each 'occp:NextRegularInspection' must occur after 'occp:UsageStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:NextRegularInspection' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

### PHASE H INSTANTS

# PHASE ASSIGNMENT

osh:PhaseHInstantShape a sh:NodeShape ;

sh:targetClass occp:PhaseH\_Instant ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseH\_SpecialEvents ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseH\_Instant' must belong to exactly one 'occp:PhaseH\_SpecialEvents' via 'occp:hasPhase'."@en ;

sh:severity sh:Violation ;

] .

### INSTANT SHAPES PHASE H - SPECIAL EVENTS

osh:GeologicalEventShape a sh:NodeShape ; # Instant Geological Event (Phase H - Special Events)

sh:targetClass occp:GeologicalEvent ;

sh:property [

sh:path occp:after ;

sh:class occp:ConstructionAcceptance ;

sh:message "Each 'occp:GeologicalEvent' must occur after 'occp:ConstructionAcceptance' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:GeologicalEvent' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:LoadEventShape a sh:NodeShape ; # Instant Load Event (Phase H - Special Events)

sh:targetClass occp:LoadEvent ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:ConstructionAcceptance ;

sh:message "Each 'occp:LoadEvent' must occur after 'occp:ConstructionAcceptance' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:LoadEvent' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:WeatherEventShape a sh:NodeShape ; # Instant Weather Event (Phase H - Special Events)

sh:targetClass occp:WeatherEvent ;

sh:property [

sh:path occp:after ;

sh:class occp:ConstructionAcceptance ;

sh:message "Each 'occp:WeatherEvent' must occur after 'occp:ConstructionAcceptance' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:WeatherEvent' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

### PHASE K INSTANTS

# PHASE ASSIGNMENT

osh:PhaseKInstantShape a sh:NodeShape ;

sh:targetClass occp:PhaseK\_Instant ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseK\_Damage ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseK\_Instant' must belong to exactly one 'occp:PhaseK\_Damage' via 'occp:hasPhase'."@en ;

sh:severity sh:Violation ;

] .

### INSTANT SHAPES PHASE K - DAMAGE

osh:DamageDetectionShape a sh:NodeShape ; # Instant Damage Detection (Phase K - Damage)

sh:targetClass occp:DamageDetection ;

sh:property [

sh:path occp:startsPhase ;

sh:class occp:PhaseK\_Damage ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:DamageDetection' must start exactly one 'occp:PhaseK\_Damage' via 'occp:startsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:ConstructionAcceptance ;

sh:message "Each 'occp:DamageDetection' must occur after 'occp:ConstructionAcceptance' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:DamageDetection' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:DamageObservationShape a sh:NodeShape ; # Instant Damage Observation (Phase K - Damage)

sh:targetClass occp:DamageObservation ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:DamageDetection ;

sh:message "Each 'occp:DamageObservation' must occur after 'occp:DamageDetection' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:DamageObservation' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:DamageRepairShape a sh:NodeShape ; # Instant Damage Repair (Phase K - Damage)

sh:targetClass occp:DamageRepair ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:DamageDetection ;

sh:message "Each 'occp:DamageRepair' must occur after 'occp:DamageDetection' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:DamageRepair' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

### PHASE L INSTANTS

# PHASE ASSIGNMENT

osh:PhaseLInstantShape a sh:NodeShape ;

sh:targetClass occp:PhaseL\_Instant ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseL\_Repair ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseL\_Instant' must belong to exactly one 'occp:PhaseL\_Repair' via 'occp:hasPhase'."@en ;

sh:severity sh:Violation ;

] .

### INSTANT SHAPES PHASE L - REPAIR

osh:NewComponentShape a sh:NodeShape ; # Instant New Component (Phase L - Repair)

sh:targetClass occp:NewComponent ;

sh:property [ # Instant's chronological order

sh:path occp:after ;

sh:class occp:ConstructionCompletion ;

sh:message "Each 'occp:NewComponent' must occur after 'occp:ConstructionCompletion' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:NewComponent' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:RepairStartShape a sh:NodeShape ; # Instant Repair Start (Phase L - Repair)

sh:targetClass occp:RepairStart ;

sh:property [ # Instant's chronological order

sh:path occp:startsPhase ;

sh:class occp:PhaseL\_Repair ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:RepairStart' must start exactly one 'occp:PhaseL\_Repair' via 'occp:startsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:after ;

sh:class occp:DamageDetection ;

sh:message "Each 'occp:RepairStart' must occur after 'occp:DamageDetection' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:RepairCompletion ;

sh:message "Each 'occp:RepairStart' must occur before 'occp:RepairCompletion' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:RepairStart' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:RepairCompletionShape a sh:NodeShape ; # Instant Repair Completion (Phase L - Repair)

sh:targetClass occp:RepairCompletion ;

sh:property [ # Instant's chronological order

sh:path occp:endsPhase ;

sh:class occp:PhaseL\_Repair ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:RepairCompletion' must end exactly one 'occp:PhaseL\_Repair' via 'occp:endsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:after ;

sh:class occp:RepairStart ;

sh:message "Each 'occp:RepairCompletion' must occur after 'occp:RepairStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:RepairCompletion' must occur before 'occp:DeconstructionStart' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

### PHASE M INSTANTS

# PHASE ASSIGNMENT

osh:PhaseMInstantShape a sh:NodeShape ;

sh:targetClass occp:PhaseM\_Instant ;

sh:property [

sh:path occp:hasPhase ;

sh:class occp:PhaseM\_Deconstruction ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseM\_Instant' must belong to exactly one 'occp:PhaseM\_Deconstruction' via 'occp:hasPhase'."@en ;

sh:severity sh:Violation ;

] .

### INSTANT SHAPES PHASE M - DECONSTRUCTION

osh:DeconstructionStartShape a sh:NodeShape ; # Instant Deconstruction Start (Phase M - Deconstruction)

sh:targetClass occp:DeconstructionStart ;

sh:property [ # Instant's chronological order

sh:path occp:startsPhase ;

sh:class occp:PhaseM\_Deconstruction ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:DeconstructionStart' must start exactly one 'occp:PhaseM\_Deconstruction' via 'occp:startsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:after ;

sh:class occp:ConstructionAcceptance ;

sh:message "Each 'occp:DeconstructionStart' must occur after 'occp:ConstructionAcceptance' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:before ;

sh:class occp:DeconstructionCompletion ;

sh:message "Each 'occp:DeconstructionStart' must occur before 'occp:DeconstructionCompletion' via 'occp:before'."@en ;

sh:severity sh:Violation ;

] .

osh:DeconstructionCompletionShape a sh:NodeShape ; # Instant Deconstruction Completion (Phase M - Deconstruction)

sh:targetClass occp:DeconstructionCompletion ;

sh:property [ # Instant's chronological order

sh:path occp:endsPhase ;

sh:class occp:PhaseM\_Deconstruction ;

sh:minCount 1 ;

sh:maxCount 1 ;

sh:message "Each 'occp:DeconstructionCompletion' must end exactly one 'occp:PhaseM\_Deconstruction' via 'occp:endsPhase'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:after ;

sh:class occp:DeconstructionStart ;

sh:message "Each 'occp:DeconstructionCompletion' must occur after 'occp:DeconstructionStart' via 'occp:after'."@en ;

sh:severity sh:Violation ;

] .

Alles klar, ich habe deine SHACL-Shapes (OCCP\_SHACL.ttl) aufgenommen und werde sie mir genau anschauen. Das ist eine beeindruckende Sammlung an Regeln, die die chronologische Konsistenz und semantische Korrektheit deiner OCCP-Ontologie sicherstellen sollen! Ich sehe, dass du sowohl allgemeine Regeln (z. B. für Zeitstempel-Formate) als auch spezifische Regeln für Phasen, Zyklen, Übergänge und Instants definiert hast – das passt perfekt zu deinem Ziel, die OCCP gründlich zu validieren.

Erster Eindruck

* Struktur: Die Shapes sind gut organisiert, mit klaren Abschnitten für allgemeine Regeln, Zyklen, Phasen und Instants. Das erleichtert die Übersicht und Wartung.
* Detailgrad: Du hast sowohl kardinalitätsbasierte Einschränkungen (z. B. sh:minCount, sh:maxCount) als auch chronologische Abhängigkeiten (z. B. occp:before, occp:after) berücksichtigt – das deckt viele potenzielle Fehlerquellen ab.
* SPARQL: Die Verwendung von SPARQL in osh:LifeCycleShape zur Prüfung von Zeitabhängigkeiten zeigt, dass du auch komplexere logische Bedingungen einbaust, was super ist.

Nächste Schritte

Um mit der Validierung der OCCP weiterzumachen, schlage ich vor, dass wir folgendes tun:

1. Konsistenzprüfung der TBox: Wir könnten die OCCP\_V0.3.ttl mit einem Reasoner (z. B. Pellet) prüfen, um sicherzustellen, dass die Ontologie an sich konsistent ist (keine logischen Widersprüche in den Klassendefinitionen oder Properties).
2. SHACL-Validierung testen: Dafür brauchen wir ein Beispiel-ABox (Instanzdaten), gegen die wir die SHACL-Shapes anwenden können. Hast du schon eine ABox-Datei (z. B. MsOCCP-Daten), die wir nutzen könnten? Falls nicht, könnten wir eine kleine Testdatei erstellen.
3. Code für Validierung: Du hast erwähnt, dass du ein Validierungsmodul hast – kannst du mir den aktuellen Stand davon zeigen? Dann könnten wir es gemeinsam anpassen, um OCCP\_V0.3.ttl und OCCP\_SHACL.ttl zu validieren (z. B. mit PySHACL oder einem anderen Tool).

Fragen an dich

* Hast du schon eine ABox (Instanzdaten), die wir testen können? Wenn ja, schick sie mir gerne!
* Welches Tool möchtest du für die SHACL-Validierung nutzen? PySHACL ist eine gängige Wahl in Python, aber wir könnten auch andere Optionen (z. B. TopBraid) betrachten.
* Gibt es bestimmte Aspekte der Validierung, auf die wir uns zuerst konzentrieren sollen (z. B. chronologische Reihenfolge, Kardinalitäten, Zyklen)?

Ich bin bereit, direkt einzusteigen – lass mich wissen, wie du vorgehen möchtest oder ob du noch etwas hochladen willst (z. B. den Validierungscode oder Testdaten)!

Hier noch das Script, das wir für die Validierung nutzen werden (validate\_shacl.py): import owlready2 from rdflib import Graph, Namespace, RDF from rdflib.namespace import SH, OWL from pyshacl import validate import logging import os import sys from io import StringIOKonfiguration des LoggingsBASE\_DIR = os.path.dirname(os.path.abspath(file)) logging.basicConfig( filename=os.path.join(BASE\_DIR, "validation.log"), level=logging.DEBUG, format="%(asctime)s - %(levelname)s - %(message)s", filemode="w" ) logger = logging.getLogger(name)Pfade und NamespaceTBOX\_PATH = os.path.join(BASE\_DIR, "OCCP\_V0.3.ttl") ABOX\_DIR = os.path.join(BASE\_DIR, "OCCP\_ABox") SHAPES\_PATH = os.path.join(BASE\_DIR, "OCCP\_SHACL.ttl") JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\", "/") OULD = Namespace("[http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD)") OCCP = Namespace("[http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#](http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod_V0.1)")def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE): try: tbox\_path\_normalized = tbox\_path.replace("\", "/") abox\_path\_normalized = abox\_path.replace("\", "/") if abox\_path else None onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle") if abox\_path: abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle") with onto: for indiv in abox\_onto.individuals(): new\_indiv = onto.get\_entities(indiv.name, indiv.class) if not new\_indiv: new\_indiv = indiv.class(indiv.name, namespace=onto) for prop in indiv.get\_properties(): for value in prop[indiv]: prop[new\_indiv] = value owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2) data\_graph = Graph() data\_graph.parse(tbox\_path, format="turtle") if abox\_path: data\_graph.parse(abox\_path, format="turtle") output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl") data\_graph.serialize(destination=output\_file, format="turtle") logger.info(f"Inferierte Ontologie gespeichert: {output\_file}") # Generische Disjunktheitsprüfung logger.info("Prüfe Ontologie auf Disjunktheit...") disjoint\_pairs = set() for s, p, o in data\_graph.triples((None, OWL.disjointWith, None)): disjoint\_pairs.add((s, o)) disjoint\_pairs.add((o, s)) # Bidirektional logger.debug(f"Disjunkte Klassenpaare: {disjoint\_pairs}") for subj in data\_graph.subjects(RDF.type, None): types = set(o for s, p, o in data\_graph.triples((subj, RDF.type, None))) for class1, class2 in disjoint\_pairs: if class1 in types and class2 in types: logger.error(f"Disjunktheitsverletzung gefunden: {subj} hat Typen {class1} und {class2}") raise Exception(f"Ontology is inconsistent: {subj} has disjoint types {class1} and {class2}") logger.info("Keine Disjunktheitsverletzungen gefunden.") return output\_file except Exception as e: logger.error(f"Fehler beim Reasoning oder Disjunktheitsprüfung: {e}") raisedef debug\_sparql(data\_file): try: data\_graph = Graph().parse(data\_file, format="turtle") query = """ PREFIX ould: [http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD) SELECT ?chain (COUNT(?u) AS ?updateCount) WHERE { ?chain a ould:UpdateChain . ?chain ould:hasUpdate ?u . } GROUP BY ?chain """ logger.info("Starte SPARQL-Abfrage für alle UpdateChains...") results = data\_graph.query(query) logger.info("SPARQL-Abfrage Ergebnisse:") for row in results: logger.info(f"Chain: {row.chain}, UpdateCount: {row.updateCount}") return len(results) > 0 except Exception as e: logger.error(f"Fehler bei der SPARQL-Abfrage: {e}") raisedef perform\_shacl\_validation(data\_file, shapes\_path=SHAPES\_PATH): try: data\_graph = Graph().parse(data\_file, format="turtle") shapes\_path\_normalized = shapes\_path.replace("\", "/") shapes\_uri = f"file:///{shapes\_path\_normalized}" logger.debug(f"Versuche Shapes von URI zu laden: {shapes\_uri}") shapes\_graph = Graph().parse(shapes\_uri, format="turtle") result = validate(data\_graph, shacl\_graph=shapes\_graph, inference="none", debug=2) conforms, report\_graph, report\_text = result logger.info(f"Konformität (inference=none): {conforms}") if not conforms: logger.info("Validierungsbericht (inference=none):") report\_lines = report\_text.splitlines() logger.info("\n".join(report\_lines)) return conforms except Exception as e: logger.error(f"Fehler bei der SHACL-Validierung: {e}") raiseif name == "main": ABOX\_PATH = os.path.join(ABOX\_DIR, "OCCP\_Valid\_LCycle\_1.ttl") inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE) debug\_sparql(inferred\_file) perform\_shacl\_validation(inferred\_file)\_\_\_\_\_ ENDE DER DATEIWir haben bereits die erste Runde der Validierung durchlaufen, die Rückmeldung war unter anderem das hier:2025-03-12 10:31:30,098 - ERROR - Fehler bei der SHACL-Validierung: '<NodeShape nf3d8ad43260341b1bcd69f6c28ebcd78b106>' exists but is not a well-formed SHACL PropertyShape. Ensure it has the correct type (sh:PropertyShape) and all required properties.Kannst Du bitte die SHACL.ttl Shapes durchgehen und prüfen, ob Du irgendwo fehlerhafte oder unvollständige Shapes findest?

Alles klar, ich habe dein validate\_shacl.py-Script und die Fehlermeldung aus dem Log aufgenommen. Die Fehlermeldung deutet darauf hin, dass es ein Problem mit einem NodeShape in deinen SHACL-Shapes gibt, das nicht korrekt als sh:PropertyShape definiert ist oder fehlerhafte/ fehlende Eigenschaften hat. Ich werde die OCCP\_SHACL.ttl-Datei durchsehen, um den Fehler zu finden, und dabei systematisch vorgehen.

Analyse der Fehlermeldung

Die Fehlermeldung lautet:

2025-03-12 10:31:30,098 - ERROR - Fehler bei der SHACL-Validierung: '<NodeShape nf3d8ad43260341b1bcd69f6c28ebcd78b106>' exists but is not a well-formed SHACL PropertyShape. Ensure it has the correct type (sh:PropertyShape) and all required properties.

* Problem: Ein NodeShape wird fälschlicherweise als sh:PropertyShape interpretiert, oder ein PropertyShape ist nicht korrekt definiert.
* Mögliche Ursachen:
  1. Ein sh:NodeShape wurde versehentlich ohne sh:property oder mit falschen Eigenschaften definiert.
  2. Ein sh:PropertyShape fehlt die obligatorische Eigenschaft sh:path oder hat einen ungültigen Typ.
  3. Syntaxfehler oder Inkonsistenzen in der TTL-Datei, die PySHACL verwirren.

Da die Fehlermeldung eine generierte Node-ID (nf3d8ad43260341b1bcd69f6c28ebcd78b106) anstelle eines benannten Shapes angibt, könnte es sich um ein implizites Shape handeln (z. B. ein verschachteltes Shape innerhalb eines sh:property-Blocks). Ich werde die Datei durchgehen und nach potenziellen Problemen suchen.

Überprüfung der OCCP\_SHACL.ttl

Ich werde die Shapes in OCCP\_SHACL.ttl systematisch prüfen, beginnend mit den allgemeinen Regeln, dann den Zyklen, Phasen und Instants. Hier ist mein Vorgehen:

1. Syntax prüfen: Sicherstellen, dass alle Shapes korrekt formatiert sind (z. B. Semikolons, Punkte).
2. NodeShapes: Überprüfen, ob jedes sh:NodeShape sinnvolle Eigenschaften hat (z. B. sh:targetClass, sh:property).
3. PropertyShapes: Sicherstellen, dass jedes sh:PropertyShape einen sh:path hat und korrekt definiert ist.
4. Verschachtelte Shapes: Prüfen, ob verschachtelte sh:property-Blöcke (z. B. mit sh:or, sh:sparql) korrekt sind.

1. Allgemeine SHACL-Regeln

* osh:GeneralDatePropertyShape:
  + Typ: sh:PropertyShape
  + sh:path: ( occp:hasActualTime occp:hasEstimatedTime ) (Listenpfad, korrekt für mehrere Properties)
  + sh:datatype: xsd:date
  + Status: Sieht korrekt aus, keine Probleme.
* osh:GeneralTimeShape:
  + Typ: sh:NodeShape
  + sh:targetClass: time:TemporalEntity
  + Verschachtelte sh:property-Blöcke:
    1. sh:path ( time:hasBeginning time:hasEnd ) mit sh:minCount 1
    2. sh:path occp:hasActualBeginning mit sh:maxCount 1
    3. sh:path occp:hasActualEnd mit sh:maxCount 1
  + Status: Korrekt definiert, die verschachtelten Shapes sind sh:PropertyShape und haben sh:path.
* osh:InstantTimeShape:
  + Typ: sh:NodeShape
  + sh:targetClass: time:Instant
  + Verschachtelte sh:property-Blöcke:
    1. sh:path occp:hasActualTime mit sh:maxCount 1
    2. sh:path occp:hasEstimatedTime (kein sh:maxCount, nur sh:datatype)
  + Potenzielles Problem: Der zweite sh:property-Block für occp:hasEstimatedTime hat keinen sh:maxCount, während occp:hasActualTime einen hat. Das könnte Inkonsistenzen verursachen, ist aber kein direkter Syntaxfehler. SHACL erfordert das nicht zwingend, also wahrscheinlich nicht die Ursache.

2. Zyklus-bezogene SHACL-Regeln

* osh:CycleShape:
  + Typ: sh:NodeShape
  + sh:targetClass: occp:Cycle
  + Verschachtelte sh:property-Blöcke: Alle haben sh:path (z. B. occp:hasCycleNumber, time:hasBeginning, time:hasEnd).
  + Status: Korrekt.
* osh:CycleAShape:
  + Typ: sh:NodeShape
  + sh:targetClass: occp:CycleA\_PlanningReview
  + Verschachtelte sh:property-Blöcke:
    1. sh:path occp:hasActualBeginning mit sh:or
    2. sh:path occp:hasActualEnd mit sh:or
    3. sh:path occp:isInPhase mit sh:or
  + Status: Die sh:or-Konstrukte sind korrekt verschachtelt, keine Probleme.
* osh:LifeCycleShape:
  + Typ: sh:NodeShape
  + sh:targetClass: occp:LifeCycle
  + Verschachtelte sh:property-Blöcke:
    1. Einschließlich eines SPARQL-basierten Blocks:

turtle

sh:sparql [

sh:select """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

PREFIX time: <http://www.w3.org/2006/time#>

SELECT $this WHERE {

$this time:hasBeginning ?start .

$this occp:isInPhase ?phase .

?phase time:hasEnd ?phaseEnd .

FILTER (?start > ?phaseEnd)

}

""" ;

sh:message "..." ;

] ;

* + Status: Der SPARQL-Block ist syntaktisch korrekt, die anderen sh:property-Blöcke ebenfalls.

3. Phasen- und Übergangs-Regeln

Ich werde nicht jede Phase einzeln durchgehen, sondern nach Mustern suchen:

* Alle osh:ShapeOfPhaseX und osh:ShapeOfTransitionX sind sh:NodeShape mit sh:targetClass.
* Viele verwenden sh:property mit sh:path und logischen Konstrukten wie sh:or, sh:and.
* Beispiel: osh:ShapeOfPhaseA:
  + sh:and mit verschachtelten Bedingungen:

turtle

sh:and (

[ sh:path occp:beginsBefore ; sh:class occp:PhaseB\_Review ]

[ sh:path occp:beginsBefore ; sh:class occp:PhaseC\_Construction ;

sh:message "..." ;

sh:severity sh:Violation ]

) ;

* + Problem: Der zweite Block innerhalb von sh:and hat ein sh:message und sh:severity, was bei einem impliziten sh:PropertyShape erlaubt ist, aber möglicherweise von PySHACL als separates Shape missverstanden wird. Laut SHACL-Spezifikation sollte das funktionieren, solange es korrekt geparst wird. Hier könnte ein Parser-Fehler vorliegen.

4. Instants

* Alle osh:PhaseXInstantShape und spezifischen Instant-Shapes (z. B. osh:BeginningOfPlanningShape) folgen dem gleichen Muster: sh:NodeShape mit sh:property-Blöcken.
* Keine offensichtlichen Syntaxfehler.

Vermutete Ursache

Nach Durchsicht der Datei sehe ich keine offensichtlichen Syntaxfehler, aber ein potenzielles Problem könnte in den verschachtelten logischen Konstrukten wie sh:and oder sh:or liegen. PySHACL könnte Schwierigkeiten haben, diese korrekt zu parsen, besonders wenn sie in einer Weise verschachtelt sind, die nicht eindeutig als sh:PropertyShape oder sh:NodeShape erkennbar ist. Ein Beispiel ist:

turtle

sh:and (

[ sh:path occp:beginsBefore ; sh:class occp:PhaseB\_Review ]

[ sh:path occp:beginsBefore ; sh:class occp:PhaseC\_Construction ;

sh:message "..." ;

sh:severity sh:Violation ]

) ;

Hier wird ein sh:and verwendet, um zwei Bedingungen zu kombinieren, aber der zweite Block hat zusätzliche Eigenschaften (sh:message, sh:severity), was PySHACL möglicherweise als separates, schlecht definiertes Shape interpretiert.

Lösungsvorschlag

1. Explizite PropertyShapes definieren: Statt verschachtelter impliziter Shapes innerhalb von sh:and, können wir sie als separate sh:PropertyShape auslagern. Beispiel für osh:ShapeOfPhaseA:

turtle

osh:ShapeOfPhaseA a sh:NodeShape ;

sh:targetClass occp:PhaseA\_Planning ;

sh:property osh:PhaseA\_BeginsBeforePhaseB ;

sh:property osh:PhaseA\_BeginsBeforePhaseC ;

sh:property [

sh:path occp:endsBefore ;

sh:class occp:PhaseC\_Construction ;

sh:message "Each 'occp:PhaseA\_Planning' must end before 'occp:PhaseC\_Construction' begins via 'occp:endsBefore'."@en ;

sh:severity sh:Warning

] ;

sh:property [

sh:path occp:endsBefore ;

sh:class occp:CompletionOfConstruction ;

sh:message "Each 'occp:PhaseA\_Planning' must end before 'occp:CompletionOfConstruction' via 'occp:endsBefore'."@en ;

sh:severity sh:Violation

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:BeginningOfPlanning ;

sh:maxCount 1 ;

sh:message "..." ;

sh:severity sh:Violation

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:or (

[ sh:class occp:ReviewApproval ]

[ sh:class occp:CompletionOfPlanning ]

) ;

sh:maxCount 1 ;

sh:message "..." ;

sh:severity sh:Violation

] .

osh:PhaseA\_BeginsBeforePhaseB a sh:PropertyShape ;

sh:path occp:beginsBefore ;

sh:class occp:PhaseB\_Review ;

sh:message "Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseB\_Review' via 'occp:beginsBefore'."@en ;

sh:severity sh:Violation .

osh:PhaseA\_BeginsBeforePhaseC a sh:PropertyShape ;

sh:path occp:beginsBefore ;

sh:class occp:PhaseC\_Construction ;

sh:message "Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'."@en ;

sh:severity sh:Violation .

Das macht die Struktur klarer und vermeidet mögliche Parser-Probleme.

1. Testlauf mit vereinfachter Datei: Wir könnten zunächst eine reduzierte Version der SHACL-Datei testen (z. B. nur die allgemeinen Regeln und eine Phase), um den Fehler einzugrenzen.
2. PySHACL-Debugging: Dein Script setzt debug=2 – das ist gut. Wir könnten zusätzlich die genaue Stelle im Report (falls vorhanden) prüfen, um den fehlerhaften Shape zu identifizieren.

Nächste Schritte

* ABox: Du hast OCCP\_Valid\_LCycle\_1.ttl erwähnt – kannst du mir diese Datei schicken? Dann kann ich die Validierung lokal reproduzieren und den Fehler genau lokalisieren.
* Fix ausprobieren: Ich kann die sh:and-Konstrukte in osh:ShapeOfPhaseA (und ähnlichen Shapes) umstellen und testen, ob das Problem dadurch behoben wird.
* Frage: Möchtest du, dass ich direkt eine überarbeitete Version der SHACL-Datei vorschlage, oder sollen wir erst mit der ABox weiter debuggen?

Lass mich wissen, wie du vorgehen willst! Ich bin bereit, den Fehler zu finden und zu beheben.

Danke Dir! Hier der INhalt der ABox (OCCP\_Valid\_LCycle\_1.ttl):

@prefix

occp: [http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#](http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod_V0.1) .

@prefix

time: [http://www.w3.org/2006/time#](http://www.w3.org/2006/time) .

@prefix

xsd: [http://www.w3.org/2001/XMLSchema#](http://www.w3.org/2001/XMLSchema) .

@prefix

: [http://www.example.de/example#](http://www.example.de/example) .Cycle und LifeCycle:cycle1 a occp:Cycle ; occp:hasCycleNumber 1 ; time:hasBeginning :instant\_cycle\_start ; time:hasEnd :instant\_cycle\_end ; occp:startsCycle :instant\_submission\_to\_review ; occp:endsCycle :instant\_review\_approval .:lifecycle1 a occp:LifeCycle ; occp:hasLifeCycleID "LC001" ; occp:isInPhase :phaseA ; time:hasBeginning :instant\_beginning\_of\_planning ; time:hasEnd :instant\_deconstruction\_completion .Phases und Transitions:phaseA a occp:PhaseA\_Planning ; occp:hasActualBeginning :instant\_beginning\_of\_planning ; occp:hasActualEnd :instant\_completion\_of\_planning ; occp:beginsBefore :phaseB ; occp:endsBefore :phaseC ; occp:startsPhase :instant\_beginning\_of\_planning ; occp:endsPhase :instant\_review\_approval .:phaseB a occp:PhaseB\_Review ; occp:hasActualBeginning :instant\_submission\_to\_review ; occp:hasActualEnd :instant\_review\_approval ; occp:beginsBefore :phaseC ; occp:startsPhase :instant\_submission\_to\_review ; occp:endsPhase :instant\_review\_approval .:transitionA a occp:CompletionOfPlanning ; occp:hasActualTime "2025-03-01"^^xsd:date ; occp:before :transitionB .:transitionB a occp:BeginOfTenderingProcess ; occp:hasActualTime "2025-03-05"^^xsd:date ; occp:before :transitionC .:transitionC a occp:Submission ; occp:hasActualTime "2025-03-10"^^xsd:date ; occp:before :phaseC .:phaseC a occp:PhaseC\_Construction ; occp:hasActualBeginning :instant\_construction\_start ; occp:hasActualEnd :instant\_completion\_of\_construction ; occp:after :transitionA ; occp:after :transitionC ; occp:startsPhase :instant\_construction\_start ; occp:endsPhase :instant\_completion\_of\_construction ; occp:hasPhase :instant\_construction\_start .:transitionD a occp:CompletionOfConstruction ; occp:hasActualTime "2025-06-01"^^xsd:date ; occp:after :transitionA ; occp:before :phaseD .:phaseD a occp:PhaseD\_Usage ; occp:hasActualBeginning :instant\_usage\_start ; occp:hasActualEnd :instant\_usage\_end ; occp:after :transitionD ; occp:endsBefore :phaseM ; occp:startsPhase :instant\_usage\_start ; occp:endsPhase :instant\_usage\_end ; occp:hasPhase :instant\_usage\_start ; occp:hasPhase :instant\_usage\_end .:phaseE a occp:PhaseE\_Warranty ; occp:hasActualBeginning :instant\_warranty\_start ; occp:hasActualEnd :instant\_warranty\_end ; occp:after :transitionD ; occp:startsPhase :instant\_warranty\_start ; occp:endsPhase :instant\_warranty\_end ; occp:hasPhase :instant\_warranty\_start ; occp:hasPhase :instant\_warranty\_end .:phaseF a occp:PhaseF\_DesignLife ; occp:hasActualBeginning :instant\_designlife\_start ; occp:hasActualEnd :instant\_designlife\_end ; occp:after :transitionD ; occp:beginsWith :phaseD ; occp:before :phaseM ; occp:startsPhase :instant\_designlife\_start ; occp:endsPhase :instant\_designlife\_end ; occp:hasPhase :instant\_designlife\_start ; occp:hasPhase :instant\_designlife\_end .:phaseG a occp:PhaseG\_Inspection ; occp:hasActualBeginning :instant\_inspection\_execution ; occp:hasActualEnd :instant\_inspection\_end ; occp:after :transitionD ; occp:before :phaseM ; occp:startsPhase :instant\_inspection\_execution ; occp:endsPhase :instant\_inspection\_end ; occp:hasPhase :instant\_inspection\_execution .:phaseH a occp:PhaseH\_SpecialEvents ; occp:hasActualBeginning :instant\_geological\_event ; occp:hasActualEnd :instant\_weather\_event ; occp:hasPhase :instant\_geological\_event ; occp:hasPhase :instant\_weather\_event .:phaseK a occp:PhaseK\_Damage ; occp:hasActualBeginning :instant\_damage\_detection ; occp:hasActualEnd :instant\_damage\_end ; occp:after :transitionC ; occp:beginsBefore :phaseL ; occp:startsPhase :instant\_damage\_detection ; occp:endsPhase :instant\_damage\_end ; occp:hasPhase :instant\_damage\_detection .:phaseL a occp:PhaseL\_Repair ; occp:hasActualBeginning :instant\_repair\_start ; occp:hasActualEnd :instant\_repair\_completion ; occp:after :transitionC ; occp:endsAfter :phaseK ; occp:before :phaseM ; occp:startsPhase :instant\_repair\_start ; occp:endsPhase :instant\_repair\_completion ; occp:hasPhase :instant\_repair\_start ; occp:hasPhase :instant\_repair\_completion .:phaseM a occp:PhaseM\_Deconstruction ; occp:hasActualBeginning :instant\_deconstruction\_start ; occp:hasActualEnd :instant\_deconstruction\_completion ; occp:after :transitionD ; occp:startsPhase :instant\_deconstruction\_start ; occp:endsPhase :instant\_deconstruction\_completion ; occp:hasPhase :instant\_deconstruction\_start ; occp:hasPhase :instant\_deconstruction\_completion .:transitionE a occp:CompletionOfDeconstruction ; occp:hasActualTime "2025-12-01"^^xsd:date ; occp:endsWith :phaseM .Instants:instant\_cycle\_start a time:Instant ; occp:hasActualTime "2025-01-01"^^xsd:date . :instant\_cycle\_end a time:Instant ; occp:hasActualTime "2025-12-01"^^xsd:date . :instant\_beginning\_of\_planning a occp:BeginningOfPlanning ; occp:hasActualTime "2025-01-01"^^xsd:date . :instant\_submission\_to\_review a occp:SubmissionToReview ; occp:hasActualTime "2025-02-15"^^xsd:date . :instant\_review\_approval a occp:ReviewApproval ; occp:hasActualTime "2025-02-28"^^xsd:date . :instant\_completion\_of\_planning a occp:CompletionOfPlanning ; occp:hasActualTime "2025-03-01"^^xsd:date . :instant\_construction\_start a occp:ConstructionStart ; occp:hasActualTime "2025-03-15"^^xsd:date . :instant\_completion\_of\_construction a occp:CompletionOfConstruction ; occp:hasActualTime "2025-06-01"^^xsd:date . :instant\_usage\_start a occp:UsageStart ; occp:hasActualTime "2025-06-02"^^xsd:date . :instant\_usage\_end a occp:UsageEnd ; occp:hasActualTime "2025-11-01"^^xsd:date . :instant\_warranty\_start a occp:WarrantyStart ; occp:hasActualTime "2025-06-02"^^xsd:date . :instant\_warranty\_end a occp:WarrantyEnd ; occp:hasActualTime "2025-11-01"^^xsd:date . :instant\_designlife\_start a occp:DesignLifeStart ; occp:hasActualTime "2025-06-02"^^xsd:date . :instant\_designlife\_end a occp:DesignLifeEnd ; occp:hasActualTime "2025-11-01"^^xsd:date . :instant\_inspection\_execution a occp:InspectionExecution ; occp:hasActualTime "2025-07-01"^^xsd:date . :instant\_inspection\_end a time:Instant ; occp:hasActualTime "2025-07-02"^^xsd:date . :instant\_geological\_event a occp:GeologicalEvent ; occp:hasActualTime "2025-08-01"^^xsd:date . :instant\_weather\_event a occp:WeatherEvent ; occp:hasActualTime "2025-09-01"^^xsd:date . :instant\_damage\_detection a occp:DamageDetection ; occp:hasActualTime "2025-07-15"^^xsd:date . :instant\_damage\_end a time:Instant ; occp:hasActualTime "2025-07-20"^^xsd:date . :instant\_repair\_start a occp:RepairStart ; occp:hasActualTime "2025-07-21"^^xsd:date . :instant\_repair\_completion a occp:RepairCompletion ; occp:hasActualTime "2025-07-25"^^xsd:date . :instant\_deconstruction\_start a occp:DeconstructionStart ; occp:hasActualTime "2025-11-15"^^xsd:date . :instant\_deconstruction\_completion a occp:CompletionOfDeconstruction ; occp:hasActualTime "2025-12-01"^^xsd:date .Vielleicht sollten wir auch, um den Fehler erstmal zu identifizieren, mit einer Minimalversion der SHACL-Shapes arbeiten bzw. doch eine kleine ABox verwenden, um zu testen, wo genau der Fehler auftaucht. Ließe sich das durch die Reduzierung der ABox regeln, oder müssen wir die SHAPES dafür ausdünnen?

Danke für die ABox (OCCP\_Valid\_LCycle\_1.ttl)! Jetzt haben wir alles, um den Fehler reproduzieren und lokalisieren zu können. Dein Vorschlag, mit einer Minimalversion der SHACL-Shapes oder einer kleineren ABox zu arbeiten, ist absolut sinnvoll, um den Fehler einzugrenzen. Ich werde beide Ansätze betrachten und dann entscheiden, wie wir am besten vorgehen.

Analyse der Situation

* Fehlermeldung: Die Fehlermeldung ('<NodeShape nf3d8ad43260341b1bcd69f6c28ebcd78b106>' exists but is not a well-formed SHACL PropertyShape) deutet darauf hin, dass PySHACL ein Problem mit einem Shape hat – wahrscheinlich ein verschachteltes sh:property oder ein logisches Konstrukt wie sh:and/sh:or, das nicht korrekt interpretiert wird.
* ABox: Die bereitgestellte ABox ist relativ umfangreich und deckt viele Phasen, Zyklen und Instants ab. Das ist gut für einen umfassenden Test, aber es macht das Debugging komplexer.
* SHACL-Shapes: Die OCCP\_SHACL.ttl enthält viele Shapes mit komplexen Strukturen (z. B. sh:and, sh:or, SPARQL), was die Wahrscheinlichkeit erhöht, dass der Fehler in einer dieser Strukturen liegt.

Strategie zur Fehlersuche

Um den Fehler effizient zu finden, schlage ich vor, schrittweise vorzugehen:

1. Minimalversion der SHACL-Shapes testen: Wir reduzieren die SHACL-Datei auf ein paar grundlegende Shapes (z. B. nur die allgemeinen Regeln und eine Phase), um zu prüfen, ob der Fehler verschwindet. Dann fügen wir schrittweise komplexere Shapes hinzu.
2. Kleine ABox erstellen: Alternativ können wir eine minimale ABox mit nur einer Phase und ein paar Instants testen, um die Datenmenge zu reduzieren.
3. Kombination: Falls nötig, kombinieren wir beides (minimale SHACL + minimale ABox), um den Fehler präzise zu lokalisieren.

Da die Fehlermeldung auf ein Problem mit den SHACL-Shapes hinweist (nicht direkt auf die ABox), fange ich mit der Reduktion der SHACL-Shapes an. Die ABox können wir später anpassen, falls der Fehler weiterhin unklar bleibt.

Schritt 1: Minimale SHACL-Shapes

Ich erstelle eine reduzierte Version von OCCP\_SHACL.ttl, die nur die allgemeinen Regeln und die Shapes für PhaseA\_Planning enthält. Das reduziert die Komplexität und hilft uns, den Fehler einzugrenzen.

Minimale OCCP\_SHACL\_min.ttl

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix osh: <http://www.occpshape.de/shape#> .

### GENERAL SHACL RULES

osh:GeneralDatePropertyShape a sh:PropertyShape ;

sh:path ( occp:hasActualTime occp:hasEstimatedTime ) ;

sh:datatype xsd:date ;

sh:message "Dates must be in YYYY-MM-DD format."@en ;

sh:severity sh:Violation .

osh:GeneralTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:property [

sh:path ( time:hasBeginning time:hasEnd ) ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:message "At least one 'time:hasBeginning' or 'time:hasEnd' must reference a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:message "Each entity must have at most one actual beginning via occp:hasActualBeginning."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:message "Each entity must have at most one actual end via occp:hasActualEnd."@en ;

sh:severity sh:Violation ;

] .

osh:InstantTimeShape a sh:NodeShape ;

sh:targetClass time:Instant ;

sh:property [

sh:path occp:hasActualTime ;

sh:datatype xsd:date ;

sh:maxCount 1 ;

sh:message "Each instant must have at most one actual timestamp via occp:hasActualTime of type xsd:date."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasEstimatedTime ;

sh:datatype xsd:date ;

sh:maxCount 1 ; # Hinzugefügt für Konsistenz

sh:message "Each instant must have at most one estimated timestamp via occp:hasEstimatedTime of type xsd:date."@en ;

sh:severity sh:Violation ;

] .

### PHASE RELATED SHACL RULES

osh:ShapeOfPhaseA a sh:NodeShape ;

sh:targetClass occp:PhaseA\_Planning ;

sh:property [

sh:path occp:endsBefore ;

sh:class occp:PhaseC\_Construction ;

sh:message "Each 'occp:PhaseA\_Planning' must end before 'occp:PhaseC\_Construction' begins via 'occp:endsBefore'."@en ;

sh:severity sh:Warning

] ;

sh:property [

sh:path occp:endsBefore ;

sh:class occp:CompletionOfConstruction ;

sh:message "Each 'occp:PhaseA\_Planning' must end before 'occp:CompletionOfConstruction' via 'occp:endsBefore'."@en ;

sh:severity sh:Violation

] ;

sh:property [

sh:path occp:beginsBefore ;

sh:class occp:PhaseB\_Review ;

sh:message "Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseB\_Review' via 'occp:beginsBefore'."@en ;

sh:severity sh:Violation

] ;

sh:property [

sh:path occp:beginsBefore ;

sh:class occp:PhaseC\_Construction ;

sh:message "Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'."@en ;

sh:severity sh:Violation

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:BeginningOfPlanning ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseA\_Planning' must have exactly one actual beginning via 'occp:hasActualBeginning' referencing 'occp:BeginningOfPlanning'."@en ;

sh:severity sh:Violation

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:or (

[ sh:class occp:ReviewApproval ]

[ sh:class occp:CompletionOfPlanning ]

) ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseA\_Planning' must have exactly one actual end via 'occp:hasActualEnd' referencing 'occp:ReviewApproval' or 'occp:CompletionOfPlanning'."@en ;

sh:severity sh:Violation

] .

Änderungen

* Entfernt: Alle anderen Phasen, Zyklen und Instants.
* Vereinfacht: Den sh:and-Block in osh:ShapeOfPhaseA durch separate sh:property-Blöcke ersetzt, um mögliche Parser-Probleme zu vermeiden.
* Ergänzt: sh:maxCount 1 bei osh:InstantTimeShape für occp:hasEstimatedTime, um Konsistenz zu gewährleisten (kein direkter Fehler, aber gute Praxis).

Schritt 2: Test mit der aktuellen ABox

Ich passe dein Script leicht an, um die minimale SHACL-Datei zu verwenden:

python

# ... (Rest des Scripts bleibt gleich)

if \_\_name\_\_ == "\_\_main\_\_":

ABOX\_PATH = os.path.join(ABOX\_DIR, "OCCP\_Valid\_LCycle\_1.ttl")

SHAPES\_PATH\_MIN = os.path.join(BASE\_DIR, "OCCP\_SHACL\_min.ttl") # Neue minimale SHACL-Datei

inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE)

debug\_sparql(inferred\_file)

perform\_shacl\_validation(inferred\_file, shapes\_path=SHAPES\_PATH\_MIN) # Minimale Shapes verwenden

Hypothese

* Wenn der Fehler verschwindet, liegt er in den entfernten Shapes (z. B. in komplexen sh:and/sh:or-Konstrukten oder SPARQL-Blöcken).
* Wenn der Fehler bleibt, können wir die ABox weiter reduzieren, um sicherzugehen, dass es nicht an den Daten liegt.

Schritt 3: Falls nötig, minimale ABox

Falls der Fehler mit der minimalen SHACL-Datei bestehen bleibt, könnten wir eine kleinere ABox wie diese testen:

turtle

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix : <http://www.example.de/example#> .

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualBeginning :instant\_beginning\_of\_planning ;

occp:hasActualEnd :instant\_review\_approval ;

occp:beginsBefore :phaseB .

:phaseB a occp:PhaseB\_Review ;

occp:hasActualBeginning :instant\_submission\_to\_review ;

occp:hasActualEnd :instant\_review\_approval .

:instant\_beginning\_of\_planning a occp:BeginningOfPlanning ;

occp:hasActualTime "2025-01-01"^^xsd:date .

:instant\_submission\_to\_review a occp:SubmissionToReview ;

occp:hasActualTime "2025-02-15"^^xsd:date .

:instant\_review\_approval a occp:ReviewApproval ;

occp:hasActualTime "2025-02-28"^^xsd:date .

Empfehlung

* Erster Test: Führe das angepasste Script mit OCCP\_SHACL\_min.ttl und der aktuellen ABox (OCCP\_Valid\_LCycle\_1.ttl) aus. Schau ins Log, ob der Fehler weg ist.
* Feedback: Teile mir das Ergebnis mit (konform oder nicht, neue Fehlermeldung falls vorhanden). Dann können wir entscheiden, ob wir die SHACL-Datei weiter erweitern oder die ABox reduzieren.

Was denkst du? Soll ich die minimale SHACL-Datei noch anpassen, oder willst du direkt mit dieser testen? Ich kann auch das Script für dich ausführen, wenn du mir sagst, wo ich die Dateien speichern soll!

Vielen Dank! Hier das Ergebnis des Tests mit der SHACL\_min.ttl und der Valid\_LCycle\_1.ttl:(Ich poste nur das, was "aus der Reihe tanzt" im Log):2025-03-12 14:46:17,107 - DEBUG - Constraint Violation in ClassConstraintComponent (<http://www.w3.org/ns/shacl#ClassConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class occp:ReviewApproval ] Focus Node: :instant\_completion\_of\_planning Value Node: :instant\_completion\_of\_planning Message: Value does not have class occp:ReviewApproval2025-03-12 14:46:17,109 - DEBUG - Constraint Violation in ClassConstraintComponent (<http://www.w3.org/ns/shacl#ClassConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class occp:CompletionOfConstruction ; sh:message Literal("Each 'occp:PhaseA\_Planning' must end before 'occp:CompletionOfConstruction' via 'occp:endsBefore'.", lang=en) ; sh:path occp:endsBefore ; sh:severity sh:Violation ] Focus Node: :phaseA Value Node: :phaseC Result Path: occp:endsBefore Message: Each 'occp:PhaseA\_Planning' must end before 'occp:CompletionOfConstruction' via 'occp:endsBefore'.2025-03-12 14:46:17,110 - DEBUG - Constraint Violation in ClassConstraintComponent (<http://www.w3.org/ns/shacl#ClassConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class occp:PhaseC\_Construction ; sh:message Literal("Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'.", lang=en) ; sh:path occp:beginsBefore ; sh:severity sh:Violation ] Focus Node: :phaseA Value Node: :phaseB Result Path: occp:beginsBefore Message: Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'.2025-03-12 14:46:17,121 - INFO - Validation Report Conforms: False Results (2): Constraint Violation in ClassConstraintComponent (<http://www.w3.org/ns/shacl#ClassConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class occp:CompletionOfConstruction ; sh:message Literal("Each 'occp:PhaseA\_Planning' must end before 'occp:CompletionOfConstruction' via 'occp:endsBefore'.", lang=en) ; sh:path occp:endsBefore ; sh:severity sh:Violation ] Focus Node: :phaseA Value Node: :phaseC Result Path: occp:endsBefore Message: Each 'occp:PhaseA\_Planning' must end before 'occp:CompletionOfConstruction' via 'occp:endsBefore'. Constraint Violation in ClassConstraintComponent (<http://www.w3.org/ns/shacl#ClassConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class occp:PhaseC\_Construction ; sh:message Literal("Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'.", lang=en) ; sh:path occp:beginsBefore ; sh:severity sh:Violation ] Focus Node: :phaseA Value Node: :phaseB Result Path: occp:beginsBefore Message: Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'.

Danke für das Log! Das ist ein großer Fortschritt – der ursprüngliche Fehler mit dem "not a well-formed SHACL PropertyShape" ist weg, was bedeutet, dass die minimale SHACL-Datei (OCCP\_SHACL\_min.ttl) syntaktisch korrekt ist und von PySHACL akzeptiert wird. Stattdessen haben wir jetzt Validierungsverletzungen, die auf logische Inkonsistenzen in der ABox (OCCP\_Valid\_LCycle\_1.ttl) hinweisen. Das ist ein gutes Zeichen, denn wir können uns jetzt auf die Daten konzentrieren und die SHACL-Regeln Schritt für Schritt wieder erweitern.

Analyse der Validierungsfehler

Ich gehe die Fehler im Log durch und erkläre, was sie bedeuten:

1. Fehler: :instant\_completion\_of\_planning hat nicht die Klasse occp:ReviewApproval

Constraint Violation in ClassConstraintComponent:

Severity: sh:Violation

Source Shape: [ sh:class occp:ReviewApproval ]

Focus Node: :instant\_completion\_of\_planning

Value Node: :instant\_completion\_of\_planning

Message: Value does not have class occp:ReviewApproval

* Quelle: osh:ShapeOfPhaseA mit:

turtle

sh:property [

sh:path occp:hasActualEnd ;

sh:or (

[ sh:class occp:ReviewApproval ]

[ sh:class occp:CompletionOfPlanning ]

) ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseA\_Planning' must have exactly one actual end via 'occp:hasActualEnd' referencing 'occp:ReviewApproval' or 'occp:CompletionOfPlanning'."@en ;

sh:severity sh:Violation

] .

* Problem: In der ABox ist :phaseA mit occp:hasActualEnd an :instant\_completion\_of\_planning gebunden:

turtle

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualEnd :instant\_completion\_of\_planning .

:instant\_completion\_of\_planning a occp:CompletionOfPlanning ;

occp:hasActualTime "2025-03-01"^^xsd:date .

Das ist an sich korrekt, denn :instant\_completion\_of\_planning hat die Klasse occp:CompletionOfPlanning, die im sh:or erlaubt ist. Der Fehler scheint hier falsch zu sein – möglicherweise ein Bug in der Berichtserstellung von PySHACL, da occp:CompletionOfPlanning akzeptiert werden sollte. Wir sollten das im Auge behalten, aber es könnte ein Anzeigefehler sein, da der Hauptbericht diesen Fehler nicht aufführt.

2. Fehler: :phaseA endet nicht vor occp:CompletionOfConstruction

Constraint Violation in ClassConstraintComponent:

Severity: sh:Violation

Source Shape: [ sh:class occp:CompletionOfConstruction ; sh:message Literal("Each 'occp:PhaseA\_Planning' must end before 'occp:CompletionOfConstruction' via 'occp:endsBefore'.", lang=en) ; sh:path occp:endsBefore ; sh:severity sh:Violation ]

Focus Node: :phaseA

Value Node: :phaseC

Result Path: occp:endsBefore

Message: Each 'occp:PhaseA\_Planning' must end before 'occp:CompletionOfConstruction' via 'occp:endsBefore'.

* Quelle: osh:ShapeOfPhaseA mit:

turtle

sh:property [

sh:path occp:endsBefore ;

sh:class occp:CompletionOfConstruction ;

sh:message "Each 'occp:PhaseA\_Planning' must end before 'occp:CompletionOfConstruction' via 'occp:endsBefore'."@en ;

sh:severity sh:Violation

] .

* Problem: In der ABox steht:

turtle

:phaseA a occp:PhaseA\_Planning ;

occp:endsBefore :phaseC .

:phaseC a occp:PhaseC\_Construction ;

occp:hasActualEnd :instant\_completion\_of\_construction .

:instant\_completion\_of\_construction a occp:CompletionOfConstruction ;

occp:hasActualTime "2025-06-01"^^xsd:date .

Die Regel verlangt, dass :phaseA via occp:endsBefore direkt mit einer Instanz von occp:CompletionOfConstruction (z. B. :instant\_completion\_of\_construction) verknüpft ist. Stattdessen ist :phaseA mit :phaseC (Typ occp:PhaseC\_Construction) verknüpft. Das ist semantisch nicht dasselbe, da occp:endsBefore eine Beziehung zu einem konkreten Ereignis (CompletionOfConstruction) erwartet, nicht zu einer Phase.

3. Fehler: :phaseA beginnt nicht vor occp:PhaseC\_Construction

Constraint Violation in ClassConstraintComponent:

Severity: sh:Violation

Source Shape: [ sh:class occp:PhaseC\_Construction ; sh:message Literal("Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'.", lang=en) ; sh:path occp:beginsBefore ; sh:severity sh:Violation ]

Focus Node: :phaseA

Value Node: :phaseB

Result Path: occp:beginsBefore

Message: Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'.

* Quelle: osh:ShapeOfPhaseA mit:

turtle

sh:property [

sh:path occp:beginsBefore ;

sh:class occp:PhaseC\_Construction ;

sh:message "Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'."@en ;

sh:severity sh:Violation

] .

* Problem: In der ABox steht:

turtle

:phaseA a occp:PhaseA\_Planning ;

occp:beginsBefore :phaseB .

:phaseB a occp:PhaseB\_Review ;

occp:beginsBefore :phaseC .

:phaseC a occp:PhaseC\_Construction .

Die Regel verlangt, dass :phaseA via occp:beginsBefore direkt mit :phaseC (Typ occp:PhaseC\_Construction) verknüpft ist. Stattdessen ist :phaseA nur mit :phaseB verknüpft. Obwohl :phaseB vor :phaseC liegt (transitive Beziehung), prüft SHACL nur direkte Verknüpfungen, es sei denn, wir verwenden Inferenz oder SPARQL, um transitive Beziehungen zu berücksichtigen.

Zusammenfassung der Probleme

1. Anzeigefehler (?): Der erste Fehler (:instant\_completion\_of\_planning) scheint ein falscher Bericht zu sein, da occp:CompletionOfPlanning erlaubt ist. Das ignorieren wir vorerst.
2. Semantischer Fehler: Die Regeln für occp:endsBefore und occp:beginsBefore erwarten direkte Verknüpfungen zu occp:CompletionOfConstruction bzw. occp:PhaseC\_Construction, aber die ABox verknüpft :phaseA mit :phaseC bzw. :phaseB. SHACL prüft nur die expliziten Triple, nicht die logische Implikation (z. B. durch Zeitstempel oder transitive Beziehungen).

Lösungsvorschläge

1. ABox anpassen

Wir könnten die ABox korrigieren, um die direkten Beziehungen herzustellen, die die SHACL-Regeln erwarten:

turtle

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualBeginning :instant\_beginning\_of\_planning ;

occp:hasActualEnd :instant\_completion\_of\_planning ;

occp:beginsBefore :phaseB ;

occp:beginsBefore :phaseC ; # Hinzugefügt

occp:endsBefore :instant\_completion\_of\_construction ; # Geändert von :phaseC

occp:startsPhase :instant\_beginning\_of\_planning ;

occp:endsPhase :instant\_review\_approval .

* Änderung 1: occp:beginsBefore :phaseC hinzufügen, damit :phaseA direkt vor :phaseC\_Construction liegt.
* Änderung 2: occp:endsBefore :phaseC durch occp:endsBefore :instant\_completion\_of\_construction ersetzen, da die Regel eine Instanz von occp:CompletionOfConstruction erwartet.

2. SHACL-Regeln anpassen

Alternativ könnten wir die SHACL-Regeln flexibler gestalten, um transitive Beziehungen oder Phasen als gültige Ziele zu akzeptieren:

* Für occp:endsBefore:

turtle

sh:property [

sh:path occp:endsBefore ;

sh:or (

[ sh:class occp:CompletionOfConstruction ]

[ sh:class occp:PhaseC\_Construction ]

) ;

sh:message "Each 'occp:PhaseA\_Planning' must end before 'occp:CompletionOfConstruction' or 'occp:PhaseC\_Construction' via 'occp:endsBefore'."@en ;

sh:severity sh:Violation

] ;

* Für occp:beginsBefore: Wir könnten eine SPARQL-Regel hinzufügen, um transitive Beziehungen zu prüfen, aber das machen wir später, wenn nötig.

3. Nächster Test

Ich empfehle, erst die ABox anzupassen, da das einfacher ist und die ursprüngliche Intention der Regeln erhält. Hier die minimale Änderung:

turtle

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualBeginning :instant\_beginning\_of\_planning ;

occp:hasActualEnd :instant\_completion\_of\_planning ;

occp:beginsBefore :phaseB ;

occp:beginsBefore :phaseC ; # Hinzugefügt

occp:endsBefore :instant\_completion\_of\_construction ; # Geändert

occp:startsPhase :instant\_beginning\_of\_planning ;

occp:endsPhase :instant\_review\_approval .

Führe das Script nochmal mit dieser geänderten ABox und OCCP\_SHACL\_min.ttl aus.

Plan

1. Testen: Passe :phaseA in der ABox wie oben an und teste erneut. Teile mir das neue Log mit!
2. Erweitern: Wenn das funktioniert (Conforms: True), können wir die SHACL-Shapes schrittweise wieder erweitern (z. B. Phase B, Zyklen) und prüfen, ob der ursprüngliche Fehler zurückkommt.
3. Debuggen: Falls der Fehler bleibt, schauen wir uns die Zeitstempel-Logik oder PySHACL-spezifische Probleme an.

Was meinst du? Soll ich die ABox für dich anpassen und dir die neue Version schicken, oder machst du das selbst und testest? Ich freue mich auf dein Feedback!

Hier der neue Report mit der geänderten PhaseA in der ABox:2025-03-12 14:55:32,142 - INFO - Validation Report Conforms: False Results (3): Validation Result in ClassConstraintComponent (<http://www.w3.org/ns/shacl#ClassConstraintComponent>): Severity: sh:Warning Source Shape: [ sh:class occp:PhaseC\_Construction ; sh:message Literal("Each 'occp:PhaseA\_Planning' must end before 'occp:PhaseC\_Construction' begins via 'occp:endsBefore'.", lang=en) ; sh:path occp:endsBefore ; sh:severity sh:Warning ] Focus Node: :phaseA Value Node: :instant\_completion\_of\_construction Result Path: occp:endsBefore Message: Each 'occp:PhaseA\_Planning' must end before 'occp:PhaseC\_Construction' begins via 'occp:endsBefore'. Constraint Violation in ClassConstraintComponent (<http://www.w3.org/ns/shacl#ClassConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class occp:PhaseB\_Review ; sh:message Literal("Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseB\_Review' via 'occp:beginsBefore'.", lang=en) ; sh:path occp:beginsBefore ; sh:severity sh:Violation ] Focus Node: :phaseA Value Node: :phaseC Result Path: occp:beginsBefore Message: Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseB\_Review' via 'occp:beginsBefore'. Constraint Violation in ClassConstraintComponent (<http://www.w3.org/ns/shacl#ClassConstraintComponent>): Severity: sh:Violation Source Shape: [ sh:class occp:PhaseC\_Construction ; sh:message Literal("Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'.", lang=en) ; sh:path occp:beginsBefore ; sh:severity sh:Violation ] Focus Node: :phaseA Value Node: :phaseB Result Path: occp:beginsBefore Message: Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'.

Danke für den neuen Report! Wir sind auf dem richtigen Weg – der ursprüngliche Syntaxfehler ist weg, und wir haben jetzt nur noch logische Validierungsfehler, die wir Schritt für Schritt lösen können. Lass uns die neuen Fehler analysieren und beheben.

Analyse des neuen Validation Reports

Der Report zeigt drei Validierungsfehler für :phaseA. Schauen wir sie uns einzeln an:

1. Fehler: :phaseA endet nicht vor occp:PhaseC\_Construction (Warning)

Validation Result in ClassConstraintComponent:

Severity: sh:Warning

Source Shape: [ sh:class occp:PhaseC\_Construction ; sh:message Literal("Each 'occp:PhaseA\_Planning' must end before 'occp:PhaseC\_Construction' begins via 'occp:endsBefore'.", lang=en) ; sh:path occp:endsBefore ; sh:severity sh:Warning ]

Focus Node: :phaseA

Value Node: :instant\_completion\_of\_construction

Result Path: occp:endsBefore

Message: Each 'occp:PhaseA\_Planning' must end before 'occp:PhaseC\_Construction' begins via 'occp:endsBefore'.

* Quelle: osh:ShapeOfPhaseA mit:

turtle

sh:property [

sh:path occp:endsBefore ;

sh:class occp:PhaseC\_Construction ;

sh:message "Each 'occp:PhaseA\_Planning' must end before 'occp:PhaseC\_Construction' begins via 'occp:endsBefore'."@en ;

sh:severity sh:Warning

] .

* ABox:

turtle

:phaseA a occp:PhaseA\_Planning ;

occp:endsBefore :instant\_completion\_of\_construction .

:instant\_completion\_of\_construction a occp:CompletionOfConstruction ;

occp:hasActualTime "2025-06-01"^^xsd:date .

:phaseC a occp:PhaseC\_Construction ;

occp:hasActualBeginning :instant\_construction\_start ;

occp:hasActualEnd :instant\_completion\_of\_construction .

:instant\_construction\_start a occp:ConstructionStart ;

occp:hasActualTime "2025-03-15"^^xsd:date .

* Problem: Die Regel erwartet, dass :phaseA via occp:endsBefore mit einer Instanz von occp:PhaseC\_Construction (z. B. :phaseC) verknüpft ist. Stattdessen ist :phaseA mit :instant\_completion\_of\_construction (Typ occp:CompletionOfConstruction) verknüpft. Das ist ein Typkonflikt, da sh:class occp:PhaseC\_Construction nicht erfüllt wird. Dies ist nur ein Warning, aber es zeigt, dass die Regel nicht das prüft, was wir vermutlich intendieren.

2. Fehler: :phaseA beginnt nicht vor occp:PhaseB\_Review

Constraint Violation in ClassConstraintComponent:

Severity: sh:Violation

Source Shape: [ sh:class occp:PhaseB\_Review ; sh:message Literal("Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseB\_Review' via 'occp:beginsBefore'.", lang=en) ; sh:path occp:beginsBefore ; sh:severity sh:Violation ]

Focus Node: :phaseA

Value Node: :phaseC

Result Path: occp:beginsBefore

Message: Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseB\_Review' via 'occp:beginsBefore'.

* Quelle: osh:ShapeOfPhaseA mit:

turtle

sh:property [

sh:path occp:beginsBefore ;

sh:class occp:PhaseB\_Review ;

sh:message "Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseB\_Review' via 'occp:beginsBefore'."@en ;

sh:severity sh:Violation

] .

* ABox:

turtle

:phaseA a occp:PhaseA\_Planning ;

occp:beginsBefore :phaseB ;

occp:beginsBefore :phaseC .

:phaseB a occp:PhaseB\_Review .

:phaseC a occp:PhaseC\_Construction .

* Problem: Die Regel verlangt, dass :phaseA via occp:beginsBefore mit einer Instanz von occp:PhaseB\_Review (z. B. :phaseB) verknüpft ist. Das ist erfüllt (occp:beginsBefore :phaseB), aber der Fehler meldet :phaseC als Value Node. Das deutet darauf hin, dass SHACL alle occp:beginsBefore-Verknüpfungen prüft und erwartet, dass jede davon den Typ occp:PhaseB\_Review hat. Die zusätzliche Verknüpfung :phaseA occp:beginsBefore :phaseC verletzt diese Regel, da :phaseC nicht vom Typ occp:PhaseB\_Review ist.

3. Fehler: :phaseA beginnt nicht vor occp:PhaseC\_Construction

Constraint Violation in ClassConstraintComponent:

Severity: sh:Violation

Source Shape: [ sh:class occp:PhaseC\_Construction ; sh:message Literal("Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'.", lang=en) ; sh:path occp:beginsBefore ; sh:severity sh:Violation ]

Focus Node: :phaseA

Value Node: :phaseB

Result Path: occp:beginsBefore

Message: Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'.

* Quelle: osh:ShapeOfPhaseA mit:

turtle

sh:property [

sh:path occp:beginsBefore ;

sh:class occp:PhaseC\_Construction ;

sh:message "Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseC\_Construction' via 'occp:beginsBefore'."@en ;

sh:severity sh:Violation

] .

* ABox:

turtle

:phaseA a occp:PhaseA\_Planning ;

occp:beginsBefore :phaseB ;

occp:beginsBefore :phaseC .

:phaseB a occp:PhaseB\_Review .

:phaseC a occp:PhaseC\_Construction .

* Problem: Ähnlich wie beim zweiten Fehler: Die Regel erwartet, dass jede occp:beginsBefore-Verknüpfung von :phaseA auf eine Instanz von occp:PhaseC\_Construction zeigt. Die Verknüpfung :phaseA occp:beginsBefore :phaseB verletzt dies, da :phaseB nicht vom Typ occp:PhaseC\_Construction ist.

Grundproblem

Die SHACL-Regeln in osh:ShapeOfPhaseA sind zu strikt formuliert:

* Jede Regel mit sh:path occp:beginsBefore oder sh:path occp:endsBefore prüft alle Werte dieser Properties und erwartet, dass sie den angegebenen Typ (sh:class) haben.
* In der ABox hat :phaseA mehrere occp:beginsBefore-Verknüpfungen (:phaseB und :phaseC), aber die Regeln erlauben nur eine spezifische Klasse pro Regel, was zu Konflikten führt.

Lösungsvorschläge

1. ABox anpassen

Wir könnten die ABox so ändern, dass :phaseA nur die erwarteten direkten Verknüpfungen hat, die den Regeln entsprechen:

turtle

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualBeginning :instant\_beginning\_of\_planning ;

occp:hasActualEnd :instant\_completion\_of\_planning ;

occp:beginsBefore :phaseB ; # Beibehalten für PhaseB\_Review

occp:beginsBefore :phaseC ; # Beibehalten für PhaseC\_Construction

occp:endsBefore :instant\_completion\_of\_construction ; # Beibehalten für CompletionOfConstruction

occp:startsPhase :instant\_beginning\_of\_planning ;

occp:endsPhase :instant\_review\_approval .

Das Problem bleibt jedoch, dass SHACL mehrere occp:beginsBefore-Werte nicht korrekt handhabt, da jede Regel nur einen Typ erwartet.

2. SHACL-Regeln anpassen (Empfohlen)

Die SHACL-Regeln sollten flexibler sein, um mehrere gültige Ziele für occp:beginsBefore und occp:endsBefore zu erlauben. Hier eine überarbeitete Version von osh:ShapeOfPhaseA:

turtle

osh:ShapeOfPhaseA a sh:NodeShape ;

sh:targetClass occp:PhaseA\_Planning ;

sh:property [

sh:path occp:endsBefore ;

sh:or (

[ sh:class occp:PhaseC\_Construction ]

[ sh:class occp:CompletionOfConstruction ]

) ;

sh:minCount 1 ; # Mindestens eine endsBefore-Beziehung

sh:message "Each 'occp:PhaseA\_Planning' must end before 'occp:PhaseC\_Construction' or 'occp:CompletionOfConstruction' via 'occp:endsBefore'."@en ;

sh:severity sh:Warning

] ;

sh:property [

sh:path occp:beginsBefore ;

sh:or (

[ sh:class occp:PhaseB\_Review ]

[ sh:class occp:PhaseC\_Construction ]

) ;

sh:minCount 1 ; # Mindestens eine beginsBefore-Beziehung

sh:message "Each 'occp:PhaseA\_Planning' must begin before 'occp:PhaseB\_Review' or 'occp:PhaseC\_Construction' via 'occp:beginsBefore'."@en ;

sh:severity sh:Violation

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:BeginningOfPlanning ;

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseA\_Planning' must have exactly one actual beginning via 'occp:hasActualBeginning' referencing 'occp:BeginningOfPlanning'."@en ;

sh:severity sh:Violation

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:or (

[ sh:class occp:ReviewApproval ]

[ sh:class occp:CompletionOfPlanning ]

) ;

 Andere Änderungen könnten sein:

sh:maxCount 1 ;

sh:message "Each 'occp:PhaseA\_Planning' must have exactly one actual end via 'occp:hasActualEnd' referencing 'occp:ReviewApproval' or 'occp:CompletionOfPlanning'."@en ;

sh:severity sh:Violation

] .

* Änderungen:
  + occp:endsBefore: Akzeptiert nun entweder occp:PhaseC\_Construction oder occp:CompletionOfConstruction.
  + occp:beginsBefore: Akzeptiert nun entweder occp:PhaseB\_Review oder occp:PhaseC\_Construction.
  + sh:minCount 1: Stellt sicher, dass mindestens eine Beziehung existiert, ohne jeden Wert einzeln zu prüfen.

3. Zeitstempel-basierte Validierung (zukünftig)

Die aktuellen Regeln prüfen nur Typen, nicht die tatsächliche zeitliche Reihenfolge (z. B. ob "2025-03-01" wirklich vor "2025-06-01" liegt). Das könnten wir später mit SPARQL-Regeln ergänzen, aber für jetzt reicht die Typprüfung.

Nächster Schritt

Ich empfehle, die SHACL-Regeln wie oben anzupassen, da das die ABox unverändert lässt und die Regeln realistischer macht. Hier die Schritte:

1. SHACL aktualisieren: Ersetze osh:ShapeOfPhaseA in OCCP\_SHACL\_min.ttl mit der neuen Version.
2. Testen: Führe das Script erneut mit der aktuellen ABox aus.
3. Feedback: Teile mir das neue Log mit – ich hoffe, wir sehen dann Conforms: True!

Frage

Soll ich dir die angepasste OCCP\_SHACL\_min.ttl direkt schicken, oder machst du die Änderung selbst? Wenn du willst, kann ich auch das Script ausführen, falls du mir sagst, wo ich die Dateien speichern soll. Lass mich wissen, wie du vorgehen möchtest!

Vielen Dank für Deine Vorschläge! Ich glaube, die ursprüngliche Logik ist an sich in Ordnung. Ich möchte nicht die Logik ändern und die Einführung von minCounts für die Beziehung halte ich auch nicht für zielführend (oder ich verstehe Deinen Ansatz nicht... Warum sollte uns das helfen?). Mein Gedanke ist, dass das Problem in der Nicht-Betrachtung der Zeitstempel liegt, die zu Fehlern oder Unklarheiten führen und ggf. in der Art die ABox aufzubauen. Beispiel: Eine Phase muss vor einer anderen Enden (Phase B muss vor Phase C sein oder zumindest vor der Transition CompletionOfPlanning). Wenn wir ein Individual in der ABox anlegen für die PhaseA, dann steht da erstmal nur drin, wann die Phase beginnt (nämlich zum Zeitpunkt BeginningOfPlanning - für diesen Instant wird ein Zeitstempel hinterlegt, der auch den Anfang von Phase A definiert. Zusätzlich müsste bei der Erstellung des PhaseA-Individuals (wegen der GeneralDatePropertyShape) mind. 1 Wert für hasEnd definiert werden - der mit einem Instant verknüpft werden muss. In diesem Fall wäre das die Transition CompletionOfPlanning, für die man dann wiederum einen Zeitstempel (am ehesten über hasEstimatedTime) hinterlegt wird. Eine Bezugsetzung der PhaseA zu anderen Phasen würde ich nicht vornehmen in der ABox, weil sich diese Bezüge ja aus der TBox und den SHACL-Rules ableiten. Die Validierung muss also nur die Verknüpfungen (also der Instant, der mit dem Anfang/Ende einer Phase verknüpft wird, muss übereinstimmen mit den jeweils zulässigen Instants für die jeweilige Phase) und die Zeitstempel überprüfen, wobei bei den Zeitstempeln immer über die zulässigen Beziehungen ausgelesen werden müsste, welche Zeitstempel miteinander zu vergleichen sind und welche die (gem. Ontologie) korrekte bzw. zulässige zeitliche Abfolge ist, um abschließend zu prüfen, ob die Zeitstempel dieser Abfolge entsprechen oder nicht. Die Abfrage der Zeitstempel haben wir noch gar nicht implementiert... Das sollten wir vielleicht tun. Was meinst Du dazu? Nun stellt sich die Frage, was passiert, wenn weitere Phasen dazukommen. Also: Wenn PhaseC eingetragen wird

Alles klar, ich verstehe deinen Ansatz jetzt besser, und du hast recht – die ursprüngliche Logik der Ontologie und SHACL-Regeln ist grundsätzlich sinnvoll, und das Problem liegt tatsächlich in der Nicht-Betrachtung der Zeitstempel sowie möglicherweise in der Art, wie die ABox aufgebaut ist. Lass uns das mal auseinandernehmen und deinen Vorschlag, die Zeitstempelprüfung einzubauen, konkret angehen.

Analyse deiner Überlegungen

1. Zeitstempel als Kern der Validierung:
   * Du möchtest, dass die Validierung nicht nur die Typen und Verknüpfungen prüft (z. B. occp:beginsBefore, occp:endsBefore), sondern auch die tatsächlichen Zeitstempel (via occp:hasActualTime oder occp:hasEstimatedTime) gegen die in der TBox/SHACL definierte zeitliche Reihenfolge validiert. Das ist absolut logisch, denn die chronologische Korrektheit ist das Herzstück deiner OCCP-Ontologie.
   * Aktuell prüfen die SHACL-Regeln nur, ob die Verknüpfungen zu den richtigen Klassen zeigen (z. B. occp:PhaseA\_Planning muss vor occp:PhaseC\_Construction beginnen), aber nicht, ob die Zeitstempel diese Reihenfolge tatsächlich widerspiegeln.
2. ABox-Aufbau:
   * Dein Punkt ist, dass die ABox primär die Phasen und ihre Start-/End-Instant mit Zeitstempeln definieren sollte (z. B. PhaseA beginnt mit BeginningOfPlanning und endet mit CompletionOfPlanning), ohne explizite Beziehungen wie occp:beginsBefore oder occp:endsBefore zu anderen Phasen direkt anzugeben. Diese Beziehungen sollen sich aus der TBox (z. B. transitive Eigenschaften von occp:before) und den SHACL-Regeln ableiten.
   * Beispiel: Wenn :phaseA mit :instant\_beginning\_of\_planning (2025-01-01) beginnt und mit :instant\_completion\_of\_planning (2025-03-01) endet, und :phaseC mit :instant\_construction\_start (2025-03-15) beginnt, sollte die Validierung prüfen, ob 2025-03-01 < 2025-03-15 ist, anstatt eine direkte occp:endsBefore-Verknüpfung zu erzwingen.
3. Warum sh:minCount nicht hilft:
   * Du hast recht, dass das Einführen von sh:minCount die Logik nicht verbessert. Mein Vorschlag war darauf abzielen, mehrere mögliche Ziele (z. B. PhaseB\_Review oder PhaseC\_Construction) zu erlauben, aber das löst nicht das eigentliche Problem der Zeitstempelprüfung. Es war eher ein Workaround für die Typkonflikte, die wir im letzten Report gesehen haben, und keine echte Lösung für die chronologische Validierung.
4. Fehlende Zeitstempelprüfung:
   * Genau wie du sagst: Wir haben die Zeitstempel noch nicht in die SHACL-Validierung integriert. Die aktuellen Regeln sind rein typbasiert und berücksichtigen nicht die Werte von occp:hasActualTime oder occp:hasEstimatedTime. Das erklärt auch, warum wir Fehler wie ":phaseA beginnt nicht vor PhaseB\_Review" bekommen – SHACL prüft nur die Typen der Verknüpfungen, nicht die zeitliche Logik.

Vorschlag: Zeitstempel-basierte Validierung implementieren

Ich stimme dir zu, dass wir die Zeitstempelprüfung einführen sollten. Das können wir mit SPARQL-basierten SHACL-Regeln machen, die die Zeitstempel vergleichen und die zulässigen Beziehungen aus der Ontologie berücksichtigen. Hier ist ein Plan, wie wir das angehen können:

1. Zeitstempel in die SHACL-Regeln integrieren

Wir ersetzen oder ergänzen die typbasierten Regeln (z. B. sh:class) durch SPARQL-Abfragen, die die Zeitstempel prüfen. Beispiel für osh:ShapeOfPhaseA:

turtle

osh:ShapeOfPhaseA a sh:NodeShape ;

sh:targetClass occp:PhaseA\_Planning ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class occp:BeginningOfPlanning ;

sh:maxCount 1 ;

sh:minCount 1 ; # Muss definiert sein wegen GeneralTimeShape

sh:message "Each 'occp:PhaseA\_Planning' must have exactly one actual beginning via 'occp:hasActualBeginning' referencing 'occp:BeginningOfPlanning'."@en ;

sh:severity sh:Violation

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:or (

[ sh:class occp:ReviewApproval ]

[ sh:class occp:CompletionOfPlanning ]

) ;

sh:maxCount 1 ;

sh:minCount 1 ; # Muss definiert sein wegen GeneralTimeShape

sh:message "Each 'occp:PhaseA\_Planning' must have exactly one actual end via 'occp:hasActualEnd' referencing 'occp:ReviewApproval' or 'occp:CompletionOfPlanning'."@en ;

sh:severity sh:Violation

] ;

sh:sparql [

sh:select """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

PREFIX time: <http://www.w3.org/2006/time#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT $this WHERE {

$this occp:hasActualEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

?phaseC a occp:PhaseC\_Construction .

?phaseC occp:hasActualBeginning ?startInstantC .

?startInstantC occp:hasActualTime ?startTimeC .

FILTER (?endTime >= ?startTimeC)

}

""" ;

sh:message "The end of 'occp:PhaseA\_Planning' (time: { ?endTime }) must be before the beginning of 'occp:PhaseC\_Construction' (time: { ?startTimeC })."@en ;

sh:severity sh:Violation

] ;

sh:sparql [

sh:select """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

PREFIX time: <http://www.w3.org/2006/time#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT $this WHERE {

$this occp:hasActualBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

?phaseB a occp:PhaseB\_Review .

?phaseB occp:hasActualBeginning ?startInstantB .

?startInstantB occp:hasActualTime ?startTimeB .

FILTER (?startTime >= ?startTimeB)

}

""" ;

sh:message "The beginning of 'occp:PhaseA\_Planning' (time: { ?startTime }) must be before the beginning of 'occp:PhaseB\_Review' (time: { ?startTimeB })."@en ;

sh:severity sh:Violation

] .

* Erklärung:
  + Die typbasierten Regeln für hasActualBeginning und hasActualEnd bleiben, um sicherzustellen, dass die Instants korrekt verknüpft sind.
  + Zwei SPARQL-Regeln prüfen die Zeitstempel:
    1. Vergleicht das Ende von PhaseA mit dem Anfang von PhaseC\_Construction.
    2. Vergleicht den Anfang von PhaseA mit dem Anfang von PhaseB\_Review.
  + FILTER stellt sicher, dass die Zeitstempel chronologisch korrekt sind (z. B. ?endTime < ?startTimeC).

2. ABox ohne direkte Beziehungen

Dein Vorschlag, die ABox ohne explizite occp:beginsBefore/occp:endsBefore-Beziehungen zu gestalten, passt dazu. Eine reduzierte ABox könnte so aussehen:

turtle

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix : <http://www.example.de/example#> .

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualBeginning :instant\_beginning\_of\_planning ;

occp:hasActualEnd :instant\_completion\_of\_planning .

:phaseB a occp:PhaseB\_Review ;

occp:hasActualBeginning :instant\_submission\_to\_review ;

occp:hasActualEnd :instant\_review\_approval .

:phaseC a occp:PhaseC\_Construction ;

occp:hasActualBeginning :instant\_construction\_start ;

occp:hasActualEnd :instant\_completion\_of\_construction .

:instant\_beginning\_of\_planning a occp:BeginningOfPlanning ;

occp:hasActualTime "2025-01-01"^^xsd:date .

:instant\_completion\_of\_planning a occp:CompletionOfPlanning ;

occp:hasActualTime "2025-03-01"^^xsd:date .

:instant\_submission\_to\_review a occp:SubmissionToReview ;

occp:hasActualTime "2025-02-15"^^xsd:date .

:instant\_review\_approval a occp:ReviewApproval ;

occp:hasActualTime "2025-02-28"^^xsd:date .

:instant\_construction\_start a occp:ConstructionStart ;

occp:hasActualTime "2025-03-15"^^xsd:date .

:instant\_completion\_of\_construction a occp:CompletionOfConstruction ;

occp:hasActualTime "2025-06-01"^^xsd:date .

* Änderung: Entferne alle occp:beginsBefore, occp:endsBefore, occp:after, etc. Die zeitliche Reihenfolge wird nur über die Zeitstempel und die SHACL-Regeln geprüft.

3. Erweiterung für weitere Phasen

Wenn :phaseC hinzukommt, müssen wir sicherstellen, dass die SPARQL-Regeln flexibel genug sind, um alle relevanten Phasen zu prüfen. Die obige Regel für PhaseC\_Construction funktioniert schon, da sie alle Instanzen von occp:PhaseC\_Construction abfragt. Für komplexere Szenarien (z. B. mehrere PhaseC-Instanzen oder transitive Beziehungen) könnten wir die Abfrage anpassen:

turtle

sh:sparql [

sh:select """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

PREFIX time: <http://www.w3.org/2006/time#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT $this WHERE {

$this occp:hasActualEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

?nextPhase occp:hasActualBeginning ?startInstantNext .

?startInstantNext occp:hasActualTime ?startTimeNext .

FILTER (?endTime >= ?startTimeNext)

FILTER (?nextPhase != $this)

FILTER (?nextPhase a occp:PhaseC\_Construction || ?nextPhase a occp:PhaseB\_Review)

}

""" ;

sh:message "The end of 'occp:PhaseA\_Planning' (time: { ?endTime }) must be before the beginning of a subsequent phase (e.g., PhaseB\_Review or PhaseC\_Construction, time: { ?startTimeNext })."@en ;

sh:severity sh:Violation

] .

* Vorteil: Prüft dynamisch alle Phasen, die nach PhaseA kommen könnten.

Nächste Schritte

1. SHACL mit Zeitstempeln testen:
   * Ich schlage vor, wir nehmen die reduzierte ABox (oben) und die neue osh:ShapeOfPhaseA mit SPARQL-Regeln und testen das mit deinem Script.
   * Das Script bleibt unverändert, da PySHACL SPARQL-Regeln nativ unterstützt.
2. Minimaltest:
   * Starte mit PhaseA, PhaseB, und PhaseC, wie in der reduzierten ABox gezeigt. Wenn das funktioniert (Conforms: True), können wir weitere Phasen hinzufügen.
3. Feedback einholen:
   * Du führst den Test durch und teilst mir das Log mit. Wenn es Probleme gibt, passen wir die SPARQL-Abfragen an.

Meine Meinung

Dein Ansatz, die Zeitstempelprüfung einzubauen, ist der richtige Weg, denn er bringt uns näher an die eigentliche Intention der OCCP: chronologische Konsistenz sicherzustellen. Die Typprüfungen allein sind zu statisch und führen zu den aktuellen Fehlern, weil sie die dynamische Natur der Phasenfolge nicht abbilden. Mit SPARQL können wir das elegant lösen.

Frage an dich

* Soll ich dir die neue OCCP\_SHACL\_min.ttl mit den SPARQL-Regeln schicken, oder möchtest du sie selbst anpassen?
* Willst du die reduzierte ABox direkt übernehmen, oder hast du eine andere Vorstellung, wie sie aussehen soll?

Lass uns das zusammen umsetzen – ich denke, wir sind kurz davor, die Validierung richtig hinzubekommen! Was meinst du?

Ok, vielen Dank! ich denke, die Einführung von SPARQL-Abfragen für die Überprüfung der Reihenfolgen anhand der Zeitstempel ist essentiell. Frage: Wäre es möglich, dass wir dafür allgemeine Regeln aufstellen statt für jede Phase individuelle Abfragen zu erstellen? Bsp.: Wenn Instanz A endeBefore Instanz B, dann: Ermittle, wann Instanz A endet (Auslesen des Zeitstempels der als Ende für Instanz A verwendeten Instanz) und wann Instanz B beginnt (analog zu Instanz A) und prüfe, ob A vor B liegt. Diese Logik können wir für alle Object properties definieren und immer wieder aufgreifen - oder?

Du hast absolut recht – die Idee, allgemeine SPARQL-Regeln für die Zeitstempelprüfung zu definieren, die auf Object Properties wie occp:endsBefore, occp:beginsBefore, etc. anwendbar sind, ist nicht nur effizient, sondern auch elegant und wiederverwendbar. Das würde die SHACL-Datei deutlich schlanker machen und die Wartbarkeit verbessern, da wir nicht für jede Phase individuelle Regeln schreiben müssen. Lass uns das konzeptuell durchdenken und umsetzen!

Konzept: Allgemeine Zeitstempel-basierte Regeln

Dein Beispiel ist ein perfekter Ausgangspunkt:

* Logik: Wenn eine Instanz A via occp:endsBefore mit Instanz B verknüpft ist, dann:
  1. Ermittle den Zeitstempel des Endes von Instanz A (über occp:hasActualEnd → occp:hasActualTime).
  2. Ermittle den Zeitstempel des Beginns von Instanz B (über occp:hasActualBeginning → occp:hasActualTime).
  3. Prüfe, ob der Endzeitpunkt von A vor dem Startzeitpunkt von B liegt (?endTimeA < ?startTimeB).

Diese Logik können wir auf alle relevanten Object Properties anwenden, die chronologische Beziehungen definieren, z. B.:

* occp:endsBefore
* occp:beginsBefore
* occp:before
* occp:after
* occp:endsAfter
* occp:beginsAfter

Vorteile

* Wiederverwendbarkeit: Eine einzige Regel kann für alle Phasen, Zyklen oder Instants gelten, solange sie diese Properties verwenden.
* Flexibilität: Funktioniert unabhängig davon, ob die ABox explizite Beziehungen (z. B. :phaseA occp:endsBefore :phaseC) enthält oder ob die Reihenfolge implizit aus den Zeitstempeln abgeleitet wird.
* Skalierbarkeit: Wenn neue Phasen oder Beziehungen hinzukommen, müssen wir nur die ABox erweitern, nicht die SHACL-Regeln.

Herausforderung

* Generalisierung: Die Regel muss so formuliert sein, dass sie sowohl direkte Beziehungen (z. B. occp:endsBefore) als auch indirekte Beziehungen (z. B. Zeitstempelvergleich ohne explizite Property) handhaben kann.
* Fehlerlokalisierung: Die Fehlermeldungen sollten spezifisch genug sein, um den Nutzer zu informieren, welche Instanzen und Zeitstempel betroffen sind.

Umsetzung: Allgemeine SHACL-Regeln mit SPARQL

Ich schlage vor, wir definieren eine generische Regel für jede der chronologischen Object Properties und binden sie an time:TemporalEntity (oder eine spezifischere Oberklasse wie occp:Phase, falls gewünscht). Hier ein Beispiel für occp:endsBefore:

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix osh: <http://www.occpshape.de/shape#> .

### Generische Zeitstempel-Regeln

osh:EndsBeforeTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ; # Gilt für alle Phasen, Zyklen, etc.

sh:sparql [

sh:select """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

PREFIX time: <http://www.w3.org/2006/time#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT $this ?other ?endTime ?startTime WHERE {

$this occp:endsBefore ?other .

$this occp:hasActualEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

?other occp:hasActualBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

FILTER (?endTime >= ?startTime)

}

""" ;

sh:message "The end of { $this } (time: { ?endTime }) must be before the beginning of { ?other } (time: { ?startTime }) via 'occp:endsBefore'."@en ;

sh:severity sh:Violation

] .

osh:BeginsBeforeTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:sparql [

sh:select """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

PREFIX time: <http://www.w3.org/2006/time#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT $this ?other ?startTime ?otherStartTime WHERE {

$this occp:beginsBefore ?other .

$this occp:hasActualBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

?other occp:hasActualBeginning ?otherStartInstant .

?otherStartInstant occp:hasActualTime ?otherStartTime .

FILTER (?startTime >= ?otherStartTime)

}

""" ;

sh:message "The beginning of { $this } (time: { ?startTime }) must be before the beginning of { ?other } (time: { ?otherStartTime }) via 'occp:beginsBefore'."@en ;

sh:severity sh:Violation

] .

### Allgemeine Struktur-Regeln (bleiben bestehen)

osh:GeneralTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:property [

sh:path ( time:hasBeginning time:hasEnd ) ;

sh:class time Maf ;

sh:minCount 1 ;

sh:message "At least one 'time:hasBeginning' or 'time:hasEnd' must reference a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:minCount 1 ; # Muss definiert sein

sh:message "Each entity must have exactly one actual beginning via occp:hasActualBeginning."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:minCount 1 ; # Muss definiert sein

sh:message "Each entity must have exactly one actual end via occp:hasActualEnd."@en ;

sh:severity sh:Violation ;

] .

osh:InstantTimeShape a sh:NodeShape ;

sh:targetClass time:Instant ;

sh:property [

sh:path occp:hasActualTime ;

sh:datatype xsd:date ;

sh:maxCount 1 ;

sh:minCount 1 ; # Muss definiert sein

sh:message "Each instant must have exactly one actual timestamp via occp:hasActualTime of type xsd:date."@en ;

sh:severity sh:Violation

] .

Erklärung

* osh:EndsBeforeTimeShape:
  + Prüft alle Instanzen, die occp:endsBefore verwenden.
  + Vergleicht den Endzeitpunkt von $this mit dem Startzeitpunkt von ?other.
  + Meldet einen Verstoß, wenn ?endTime >= ?startTime.
* osh:BeginsBeforeTimeShape:
  + Prüft alle Instanzen mit occp:beginsBefore.
  + Vergleicht den Startzeitpunkt von $this mit dem Startzeitpunkt von ?other.
* Allgemeine Regeln: Stellen sicher, dass jede Phase/Instanz die notwendigen Zeitstempel hat (minCount 1).

ABox-Beispiel

turtle

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix : <http://www.example.de/example#> .

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualBeginning :instant\_beginning\_of\_planning ;

occp:hasActualEnd :instant\_completion\_of\_planning ;

occp:endsBefore :phaseC .

:phaseB a occp:PhaseB\_Review ;

occp:hasActualBeginning :instant\_submission\_to\_review ;

occp:hasActualEnd :instant\_review\_approval ;

occp:beginsBefore :phaseC .

:phaseC a occp:PhaseC\_Construction ;

occp:hasActualBeginning :instant\_construction\_start ;

occp:hasActualEnd :instant\_completion\_of\_construction .

:instant\_beginning\_of\_planning a occp:BeginningOfPlanning ;

occp:hasActualTime "2025-01-01"^^xsd:date .

:instant\_completion\_of\_planning a occp:CompletionOfPlanning ;

occp:hasActualTime "2025-03-01"^^xsd:date .

:instant\_submission\_to\_review a occp:SubmissionToReview ;

occp:hasActualTime "2025-02-15"^^xsd:date .

:instant\_review\_approval a occp:ReviewApproval ;

occp:hasActualTime "2025-02-28"^^xsd:date .

:instant\_construction\_start a occp:ConstructionStart ;

occp:hasActualTime "2025-03-15"^^xsd:date .

:instant\_completion\_of\_construction a occp:CompletionOfConstruction ;

occp:hasActualTime "2025-06-01"^^xsd:date .

* Test: Diese ABox sollte Conforms: True ergeben, da "2025-03-01" < "2025-03-15" (für endsBefore) und "2025-02-15" < "2025-03-15" (für beginsBefore).

Erweiterung für andere Properties

Wir können ähnliche Regeln für occp:before, occp:after, etc. erstellen:

* occp:before:

turtle

osh:BeforeTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:sparql [

sh:select """

SELECT $this ?other ?endTime ?startTime WHERE {

$this occp:before ?other .

$this occp:hasActualEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

?other occp:hasActualBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

FILTER (?endTime >= ?startTime)

}

""" ;

sh:message "The end of { $this } (time: { ?endTime }) must be before the beginning of { ?other } (time: { ?startTime }) via 'occp:before'."@en ;

sh:severity sh:Violation

] .

Ohne explizite Beziehungen

Falls die ABox keine occp:endsBefore/occp:beginsBefore enthält, könnten wir eine Regel schreiben, die die implizite Reihenfolge prüft:

turtle

osh:PhaseOrderTimeShape a sh:NodeShape ;

sh:targetClass occp:Phase ;

sh:sparql [

sh:select """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

PREFIX time: <http://www.w3.org/2006/time#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT $this ?other ?endTime ?startTime WHERE {

$this occp:hasActualEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

?other occp:hasActualBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

FILTER (?endTime >= ?startTime)

FILTER (?other != $this)

FILTER (

($this a occp:PhaseA\_Planning && ?other a occp:PhaseC\_Construction) ||

($this a occp:PhaseB\_Review && ?other a occp:PhaseC\_Construction)

)

}

""" ;

sh:message "The end of { $this } (time: { ?endTime }) must be before the beginning of { ?other } (time: { ?startTime }) based on phase order."@en ;

sh:severity sh:Violation

] .

* Hinweis: Hier müssen wir die erlaubten Reihenfolgen (z. B. A vor C, B vor C) explizit im FILTER auflisten, was weniger generisch ist.

Nächste Schritte

1. Minimalversion testen:
   * Nutze die oben gezeigten osh:EndsBeforeTimeShape und osh:BeginsBeforeTimeShape in einer neuen OCCP\_SHACL\_min.ttl.
   * Teste mit der ABox, die occp:endsBefore und occp:beginsBefore enthält.
2. Erweitern:
   * Füge Regeln für occp:before, occp:after, etc. hinzu.
   * Optional: Teste eine ABox ohne explizite Beziehungen mit der impliziten Regel.
3. Feedback:
   * Du führst den Test durch und teilst mir das Log mit.

Antwort auf deine Frage

Ja, wir können absolut allgemeine Regeln für alle Object Properties definieren! Die SPARQL-basierte Lösung oben ist genau dafür gemacht. Sie ist wiederverwendbar und kann auf jede Property angewendet werden, die eine chronologische Beziehung definiert. Die Herausforderung ohne explizite Beziehungen lösen wir entweder durch eine implizite Regel (mit festgelegter Reihenfolge) oder durch Reasoning in der TBox (z. B. transitive Eigenschaften), aber für den Anfang reicht die explizite Variante.

Vorschlag

Ich schicke dir eine OCCP\_SHACL\_min.ttl mit den generischen Regeln für endsBefore und beginsBefore. Du kannst sie mit deiner ABox testen. Soll ich das machen, oder willst du die Regeln selbst schreiben? Wie siehst du das?

Vielen Dank! In Deinem Codevorschlag für die SHACL-Regeln mit SPARQL steht u.a. folgende Zeile: sh:class time Maf ;Dafür habe ich sh:class time:Instant ; eingesetzt. Der Test mit Deiner angepassten, minimalen ABox lief durch (vielen Dank für Deine Vorschläge, sehr wertvoll!!) und das Ergebnis ist:2025-03-12 17:22:19,631 - INFO - Konformit�t (inference=none): True:-)Yeah! Danke!Aus meiner Sicht dürfte in der ABox bei PhaseA und B nicht occp:endsBefore :phaseC drinstehen, denn das ist überflüssig (weil die Reihenfolge von den Verknüpfungen mit den Instanzen definiert wird und die TBox die zulässigen Verknüpfungen und die zulässige Reihenfolge (im Bezug auf die Zeitstempel) vorgibt). Ich habe die beiden Angaben mal rausgelöscht und das Ergebnis der Validierung war (erwartungsgemäß) dasselbe. Aber es ist wichtig, dass die ABoxen nur die Informationen enthalten, die sie enthalten müssen, um eine vollständige Chronologie herstellen zu können.Wie wollen wir weitermachen?

Super, das freut mich riesig – Conforms: True ist ein großer Erfolg! Und danke fürs Korrigieren von sh:class time Maf zu sh:class time:Instant – das war definitiv ein Tippfehler meinerseits (wahrscheinlich ein Copy-Paste-Fehler). Schön, dass der Test mit der minimalen ABox durchgelaufen ist und auch ohne die expliziten occp:endsBefore-Angaben funktioniert hat. Dein Punkt zur ABox ist absolut richtig, und ich stimme dir zu, dass wir jetzt eine solide Basis haben, um weiterzumachen.

Analyse des aktuellen Standes

* Zeitstempelprüfung funktioniert: Die SPARQL-Regeln prüfen die chronologische Reihenfolge korrekt anhand der Zeitstempel, wenn occp:endsBefore oder occp:beginsBefore vorhanden sind.
* Redundanz in der ABox: Du hast vollkommen recht, dass occp:endsBefore :phaseC in :phaseA und :phaseB überflüssig ist, solange die Zeitstempel und die Verknüpfungen zu den Instants (z. B. :instant\_completion\_of\_planning) die Reihenfolge definieren. Die TBox und SHACL-Regeln sollten die erlaubten Verknüpfungen und die chronologische Logik abdecken, während die ABox nur die minimal notwendigen Daten liefert (Phasen + Start/End-Instant mit Zeitstempeln).
* Minimalistische ABox: Dein Test ohne diese Beziehungen zeigt, dass wir die ABox schlank halten können, was die Datenpflege erleichtert und Redundanzen vermeidet.

Dein Punkt zur ABox-Philosophie

Ich unterstütze deine Sichtweise voll und ganz:

* Die ABox sollte nur die Rohdaten enthalten: Phasen, ihre Start- und End-Instant sowie deren Zeitstempel.
* Die Reihenfolge und Constraints (z. B. "Phase A muss vor Phase C enden") sollten aus der TBox (Ontologie-Definitionen) und den SHACL-Regeln abgeleitet werden.
* Das bedeutet, dass wir die SHACL-Regeln so anpassen müssen, dass sie die chronologische Reihenfolge auch ohne explizite Beziehungen wie occp:endsBefore prüfen können – basierend auf den Typen der Phasen und den Zeitstempeln.

Wie weitermachen?

Wir haben jetzt zwei Optionen, wie wir die Validierung erweitern können, um deine Vision vollständig umzusetzen:

1. SHACL-Regeln ohne explizite Beziehungen (implizite Reihenfolge)

Da du die occp:endsBefore/occp:beginsBefore-Angaben aus der ABox entfernen möchtest, müssen wir die SHACL-Regeln so umstellen, dass sie die Reihenfolge basierend auf den Phasen-Typen und Zeitstempeln prüfen, ohne auf direkte Beziehungen angewiesen zu sein. Beispiel:

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix osh: <http://www.occpshape.de/shape#> .

### Allgemeine Struktur-Regeln

osh:GeneralTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:property [

sh:path ( time:hasBeginning time:hasEnd ) ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:message "At least one 'time:hasBeginning' or 'time:hasEnd' must reference a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each entity must have exactly one actual beginning via occp:hasActualBeginning."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each entity must have exactly one actual end via occp:hasActualEnd."@en ;

sh:severity sh:Violation ;

] .

osh:InstantTimeShape a sh:NodeShape ;

sh:targetClass time:Instant ;

sh:property [

sh:path occp:hasActualTime ;

sh:datatype xsd:date ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each instant must have exactly one actual timestamp via occp:hasActualTime of type xsd:date."@en ;

sh:severity sh:Violation

] .

### Implizite Reihenfolge-Regel

osh:PhaseOrderTimeShape a sh:NodeShape ;

sh:targetClass occp:Phase ;

sh:sparql [

sh:select """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

PREFIX time: <http://www.w3.org/2006/time#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT $this ?other ?endTime ?startTime WHERE {

$this occp:hasActualEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

?other occp:hasActualBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

FILTER (?endTime >= ?startTime)

FILTER (?other != $this)

FILTER (

($this a occp:PhaseA\_Planning && ?other a occp:PhaseB\_Review) ||

($this a occp:PhaseA\_Planning && ?other a occp:PhaseC\_Construction) ||

($this a occp:PhaseB\_Review && ?other a occp:PhaseC\_Construction)

)

}

""" ;

sh:message "The end of { $this } (time: { ?endTime }) must be before the beginning of { ?other } (time: { ?startTime }) based on phase order."@en ;

sh:severity sh:Violation

] .

* ABox-Test:

turtle

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix : <http://www.example.de/example#> .

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualBeginning :instant\_beginning\_of\_planning ;

occp:hasActualEnd :instant\_completion\_of\_planning .

:phaseB a occp:PhaseB\_Review ;

occp:hasActualBeginning :instant\_submission\_to\_review ;

occp:hasActualEnd :instant\_review\_approval .

:phaseC a occp:PhaseC\_Construction ;

occp:hasActualBeginning :instant\_construction\_start ;

occp:hasActualEnd :instant\_completion\_of\_construction .

:instant\_beginning\_of\_planning a occp:BeginningOfPlanning ;

occp:hasActualTime "2025-01-01"^^xsd:date .

:instant\_completion\_of\_planning a occp:CompletionOfPlanning ;

occp:hasActualTime "2025-03-01"^^xsd:date .

:instant\_submission\_to\_review a occp:SubmissionToReview ;

occp:hasActualTime "2025-02-15"^^xsd:date .

:instant\_review\_approval a occp:ReviewApproval ;

occp:hasActualTime "2025-02-28"^^xsd:date .

:instant\_construction\_start a occp:ConstructionStart ;

occp:hasActualTime "2025-03-15"^^xsd:date .

:instant\_completion\_of\_construction a occp:CompletionOfConstruction ;

occp:hasActualTime "2025-06-01"^^xsd:date .

* Erwartung: Conforms: True, da die Zeitstempel die Reihenfolge A → B → C einhalten.

2. Vollständige SHACL-Datei erweitern

Sobald die implizite Regel funktioniert, können wir die vollständige OCCP\_SHACL.ttl wieder einbeziehen und:

* Die typbasierten Regeln (z. B. sh:class) beibehalten, um die erlaubten Instants pro Phase zu prüfen (z. B. PhaseA endet mit ReviewApproval oder CompletionOfPlanning).
* Die SPARQL-Regeln für die Zeitstempelprüfung hinzufügen, entweder generisch (wie oben) oder spezifisch für bestimmte Phasen, falls du doch mehr Granularität willst.

Mein Vorschlag

* Schritt 1: Teste die obige osh:PhaseOrderTimeShape mit der reduzierten ABox ohne explizite Beziehungen. Das validiert deine Idee, die ABox minimalistisch zu halten.
* Schritt 2: Wenn das klappt, erweitern wir die SPARQL-Regel, um alle Phasen aus deiner ursprünglichen Ontologie abzudecken (z. B. Phase D, E, F, etc.) – das FILTER wird dann länger, aber bleibt generisch.
* Schritt 3: Integriere die generische Regel in die volle OCCP\_SHACL.ttl und teste mit deiner kompletten ABox (OCCP\_Valid\_LCycle\_1.ttl), natürlich ohne die redundanten Beziehungen.

Wie vorgehen?

* Ich kann dir die OCCP\_SHACL\_min.ttl mit der impliziten osh:PhaseOrderTimeShape-Regel schicken, und du testest sie mit der reduzierten ABox.
* Alternativ kannst du die Regel selbst in deine aktuelle OCCP\_SHACL\_min.ttl einfügen und testen.

Was meinst du? Soll ich dir die Datei schicken, oder gehst du direkt ran? Und wie siehst du die Erweiterung auf die volle Ontologie – sollen wir das gleich angehen oder erst die implizite Regel perfektionieren? Ich freue mich auf dein Feedback – wir sind echt gut vorangekommen!

Sehr gut, danke! Ja, lass uns die generischen SPARQL-Abfragen erstellen, einbauen und testen. Ich denke, wenn wir die hinbekommen, dann reicht das erstmal für ein Minimalbeispiel aus (und wenn ich später die Zeit finde, kann man die generischen Regeln aufgreifen und an die einzelnen Phasen und ggf. Instants individuell anpassen). Wichtig ist, dass wir keine Object Property (OP) vergessen (bei der Erstellung unserer SPARQL-Abfragen (SA)). Ich denke, wir werden wahrscheinlich auch eine SA für time:after und time:before (und ggf. weitere time:OPs) benötigen (nicht nur für die ganzen occp:OPs). Aber eins nach dem anderen. Lass uns mit OCCP after und before beginnen, oder was sagst Du?

Alles klar, ich bin dabei! Dein Plan, mit generischen SPARQL-Abfragen (SAs) zu starten und sie später bei Bedarf an spezifische Phasen oder Instants anzupassen, ist perfekt. Und du hast recht – wir müssen sicherstellen, dass wir keine relevante Object Property (OP) vergessen, sowohl aus dem occp:-Namespace als auch aus time:, falls sie in der Ontologie verwendet werden. Ich stimme dir zu, dass wir mit occp:after und occp:before beginnen sollten, da das die Kernbeziehungen für die chronologische Reihenfolge sind. Danach können wir schrittweise erweitern (z. B. time:after, time:before, und andere occp:-OPs wie endsBefore, beginsBefore).

Strategie

* Ziel: Generische SAs für occp:after und occp:before, die Zeitstempel prüfen, unabhängig von spezifischen Phasen.
* Minimalbeispiel: Wir testen sie mit einer kleinen ABox, um sicherzugehen, dass sie funktionieren.
* Erweiterung: Später fügen wir SAs für weitere OPs hinzu (z. B. time:after, time:before, occp:endsBefore) und prüfen, ob wir sie anpassen müssen.

Generische SPARQL-Regeln für occp:after und occp:before

Hier sind die SAs für occp:after und occp:before, die auf time:TemporalEntity angewendet werden, um maximale Generizität zu gewährleisten:

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix osh: <http://www.occpshape.de/shape#> .

### Allgemeine Struktur-Regeln

osh:GeneralTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:property [

sh:path ( time:hasBeginning time:hasEnd ) ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:message "At least one 'time:hasBeginning' or 'time:hasEnd' must reference a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each entity must have exactly one actual beginning via occp:hasActualBeginning."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each entity must have exactly one actual end via occp:hasActualEnd."@en ;

sh:severity sh:Violation ;

] .

osh:InstantTimeShape a sh:NodeShape ;

sh:targetClass time:Instant ;

sh:property [

sh:path occp:hasActualTime ;

sh:datatype xsd:date ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each instant must have exactly one actual timestamp via occp:hasActualTime of type xsd:date."@en ;

sh:severity sh:Violation

] .

### Generische Zeitstempel-Regeln für occp:after

osh:AfterTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:sparql [

sh:select """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

PREFIX time: <http://www.w3.org/2006/time#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT $this ?other ?startTime ?endTime WHERE {

$this occp:after ?other .

$this occp:hasActualBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

?other occp:hasActualEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

FILTER (?startTime <= ?endTime)

}

""" ;

sh:message "The beginning of { $this } (time: { ?startTime }) must be after the end of { ?other } (time: { ?endTime }) via 'occp:after'."@en ;

sh:severity sh:Violation

] .

### Generische Zeitstempel-Regeln für occp:before

osh:BeforeTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:sparql [

sh:select """

PREFIX occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#>

PREFIX time: <http://www.w3.org/2006/time#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT $this ?other ?endTime ?startTime WHERE {

$this occp:before ?other .

$this occp:hasActualEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

?other occp:hasActualBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

FILTER (?endTime >= ?startTime)

}

""" ;

sh:message "The end of { $this } (time: { ?endTime }) must be before the beginning of { ?other } (time: { ?startTime }) via 'occp:before'."@en ;

sh:severity sh:Violation

] .

Erklärung

* osh:AfterTimeShape:
  + Prüft: Wenn $this occp:after ?other, dann muss der Start von $this nach dem Ende von ?other liegen (?startTime > ?endTime).
  + Fehler: Wenn ?startTime <= ?endTime, wird ein Verstoß gemeldet.
* osh:BeforeTimeShape:
  + Prüft: Wenn $this occp:before ?other, dann muss das Ende von $this vor dem Start von ?other liegen (?endTime < ?startTime).
  + Fehler: Wenn ?endTime >= ?startTime, wird ein Verstoß gemeldet.
* Struktur-Regeln: Stellen sicher, dass jede Phase/Instanz die notwendigen Zeitstempel hat.

Test-ABox

Um das zu testen, nehmen wir eine kleine ABox mit expliziten occp:after und occp:before-Beziehungen:

turtle

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix : <http://www.example.de/example#> .

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualBeginning :instant\_beginning\_of\_planning ;

occp:hasActualEnd :instant\_completion\_of\_planning ;

occp:before :phaseB .

:phaseB a occp:PhaseB\_Review ;

occp:hasActualBeginning :instant\_submission\_to\_review ;

occp:hasActualEnd :instant\_review\_approval ;

occp:after :phaseA ;

occp:before :phaseC .

:phaseC a occp:PhaseC\_Construction ;

occp:hasActualBeginning :instant\_construction\_start ;

occp:hasActualEnd :instant\_completion\_of\_construction ;

occp:after :phaseB .

:instant\_beginning\_of\_planning a occp:BeginningOfPlanning ;

occp:hasActualTime "2025-01-01"^^xsd:date .

:instant\_completion\_of\_planning a occp:CompletionOfPlanning ;

occp:hasActualTime "2025-03-01"^^xsd:date .

:instant\_submission\_to\_review a occp:SubmissionToReview ;

occp:hasActualTime "2025-02-15"^^xsd:date .

:instant\_review\_approval a occp:ReviewApproval ;

occp:hasActualTime "2025-02-28"^^xsd:date .

:instant\_construction\_start a occp:ConstructionStart ;

occp:hasActualTime "2025-03-15"^^xsd:date .

:instant\_completion\_of\_construction a occp:CompletionOfConstruction ;

occp:hasActualTime "2025-06-01"^^xsd:date .

* Erwartung: Conforms: True, da:
  + PhaseA (Ende: 2025-03-01) vor PhaseB (Start: 2025-02-15) → Fehler, weil 2025-03-01 > 2025-02-15!
  + PhaseB (Start: 2025-02-15) nach PhaseA (Ende: 2025-03-01) → Fehler, weil 2025-02-15 < 2025-03-01!
  + PhaseC (Start: 2025-03-15) nach PhaseB (Ende: 2025-02-28) → OK, weil 2025-03-15 > 2025-02-28.

Moment – die ABox hat hier ein Problem! Die Zeitstempel sind inkonsistent mit den Beziehungen:

* PhaseA occp:before PhaseB verlangt, dass 2025-03-01 < 2025-02-15 → falsch.
* PhaseB occp:after PhaseA verlangt, dass 2025-02-15 > 2025-03-01 → falsch.

Korrigierte ABox

turtle

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualBeginning :instant\_beginning\_of\_planning ;

occp:hasActualEnd :instant\_completion\_of\_planning ;

occp:before :phaseB .

:phaseB a occp:PhaseB\_Review ;

occp:hasActualBeginning :instant\_submission\_to\_review ;

occp:hasActualEnd :instant\_review\_approval ;

occp:after :phaseA ;

occp:before :phaseC .

:phaseC a occp:PhaseC\_Construction ;

occp:hasActualBeginning :instant\_construction\_start ;

occp:hasActualEnd :instant\_completion\_of\_construction ;

occp:after :phaseB .

:instant\_beginning\_of\_planning a occp:BeginningOfPlanning ;

occp:hasActualTime "2025-01-01"^^xsd:date .

:instant\_completion\_of\_planning a occp:CompletionOfPlanning ;

occp:hasActualTime "2025-02-01"^^xsd:date . # Geändert

:instant\_submission\_to\_review a occp:SubmissionToReview ;

occp:hasActualTime "2025-02-15"^^xsd:date .

:instant\_review\_approval a occp:ReviewApproval ;

occp:hasActualTime "2025-02-28"^^xsd:date .

:instant\_construction\_start a occp:ConstructionStart ;

occp:hasActualTime "2025-03-15"^^xsd:date .

:instant\_completion\_of\_construction a occp:CompletionOfConstruction ;

occp:hasActualTime "2025-06-01"^^xsd:date .

* Jetzt:
  + PhaseA (Ende: 2025-02-01) vor PhaseB (Start: 2025-02-15) → OK.
  + PhaseB (Start: 2025-02-15) nach PhaseA (Ende: 2025-02-01) → OK.
  + PhaseC (Start: 2025-03-15) nach PhaseB (Ende: 2025-02-28) → OK.
  + Erwartung: Conforms: True.

Nächste Schritte

1. Testen:
   * Nutze die obige SHACL-Datei (OCCP\_SHACL\_min.ttl) mit den SAs für occp:after und occp:before.
   * Teste mit der korrigierten ABox.
2. Erweitern:
   * Füge SAs für weitere occp:-OPs hinzu, z. B.:
     + occp:endsBefore
     + occp:beginsBefore
     + occp:endsAfter
     + occp:beginsAfter
   * Danach können wir time:after und time:before ergänzen, falls sie in deiner TBox verwendet werden.
3. Feedback:
   * Du führst den Test durch und teilst mir das Log mit. Wenn es Probleme gibt, passen wir die SAs an.

Meine Meinung

Ich denke, mit occp:after und occp:before zu starten ist ideal, da sie die grundlegendsten chronologischen Beziehungen abdecken. Sobald wir das stabil haben, können wir die anderen OPs schrittweise einbauen. Dein Hinweis auf time:after und time:before ist wichtig – wir sollten checken, ob sie in OCCP\_V0.3.ttl vorkommen, und sie dann bei Bedarf ergänzen.

Soll ich dir die OCCP\_SHACL\_min.ttl mit den beiden SAs schicken, oder passt du sie selbst an? Und wie siehst du die Reihenfolge für die weiteren OPs? Lass uns das rocken!

Vielen Dank! Müssen wir die PREFIXes in jeder SA von Neuem angeben? Vor allem, wenn diese bereits in der TTL-Datei am Anfang definiert wurden? Und: Du nutzt in Deiner Abfrage Actual-Zeiten - sollten wir da vielleicht (um es maximal generisch zu gestalten) die übergeordneten OPs (also z.B. hasBeginning anstelle von hasActualBeginning) nutzen oder spricht da aus Deiner Sicht etwas dagegen? Deine korrigierte ABox ist an mehreren Stellen falsch. Du hast wieder zusätzliche Beziehungen (occp:before :phaseB) gemacht, die wir nicht brauchen. Außerdem hast Du den Zeitpunkt für CompletionOfPlanning VOR SubmissionToReview gelegt, das wäre auch nicht möglich. Bitte prüf das noch einmal und korrigiere es.

Danke für dein Feedback! Du hast vollkommen recht – ich werde deine Punkte Schritt für Schritt adressieren und die Vorschläge entsprechend anpassen. Lass uns das klären und korrigieren!

1. PREFIXes in SPARQL-Abfragen

* Frage: Müssen wir die PREFIXes in jeder SA neu angeben, wenn sie schon am Anfang der TTL-Datei definiert sind?
* Antwort: Nein, das ist nicht nötig! Die PREFIX-Definitionen am Anfang der TTL-Datei gelten für die gesamte Datei, einschließlich aller eingebetteten SPARQL-Abfragen. SHACL (und RDF im Allgemeinen) übernimmt die Namespace-Definitionen aus dem Dokument-Kontext. Ich hatte die PREFIXes in den SAs aus Gewohnheit und für Lesbarkeit wiederholt, aber wir können sie weglassen, da sie redundant sind. Ich werde das in der überarbeiteten Version korrigieren.

2. Verwendung von hasActualBeginning vs. time:hasBeginning

* Dein Punkt: Sollten wir generischere OPs wie time:hasBeginning statt occp:hasActualBeginning verwenden, um maximale Flexibilität zu gewährleisten?
* Meine Sicht: Das ist eine ausgezeichnete Idee! time:hasBeginning und time:hasEnd sind Teil der W3C TIME-Ontologie und übergeordnet zu den spezifischen occp:-Properties wie hasActualBeginning und hasActualEnd. Wenn wir time:hasBeginning/time:hasEnd verwenden:
  + Vorteile:
    - Generischer: Funktioniert auch, wenn jemand zusätzlich zu hasActualBeginning andere Subproperties definiert (z. B. hasEstimatedBeginning).
    - Konsistenter mit der TIME-Ontologie, die deine OCCP erweitert.
    - Weniger Abhängigkeit von spezifischen occp:-Properties, was die Regeln robuster macht.
  + Gegenargumente:
    - Wenn deine Ontologie strikt nur occp:hasActualBeginning/occp:hasActualEnd erlaubt (z. B. durch Domain/Range-Einschränkungen oder SHACL-Regeln), könnten wir uns darauf beschränken, um spezifischer zu sein.
    - time:hasBeginning könnte mehrere Instants erlauben (kein maxCount 1), während hasActualBeginning durch die bestehenden SHACL-Regeln auf genau einen Instant beschränkt ist.
  + Fazit: Da wir generische Regeln wollen und deine osh:GeneralTimeShape schon sicherstellt, dass time:hasBeginning/time:hasEnd mindestens einen Instant haben (und hasActualBeginning/hasActualEnd zusätzlich maxCount 1 erzwingen), können wir time:hasBeginning/time:hasEnd verwenden. Das macht die SAs flexibler und zukunftssicher. Ich passe das an!

3. Korrektur der ABox

* Deine Kritik:
  1. Ich habe unnötige Beziehungen wie occp:before :phaseB hinzugefügt, obwohl wir die ABox minimalistisch halten wollen.
  2. Der Zeitpunkt für instant\_completion\_of\_planning (2025-02-01) liegt vor instant\_submission\_to\_review (2025-02-15), was chronologisch nicht zulässig ist, da CompletionOfPlanning nach SubmissionToReview kommen muss.
* Analyse:
  1. Überflüssige Beziehungen: Mea culpa – ich wollte die Beziehungen nur für den Test explizit machen, aber das widerspricht unserem Ziel, die ABox schlank zu halten. Ich entferne sie komplett.
  2. Zeitstempel-Fehler: Du hast recht, die Reihenfolge ist falsch. In deiner Ontologie muss SubmissionToReview (Start von Phase B) vor CompletionOfPlanning (Ende von Phase A) liegen, da die Planungsphase (A) erst mit der Fertigstellung nach der Überprüfung (B) endet. Ich hatte die Zeitstempel vertauscht, um die after-Regel zu testen, ohne die Logik deiner Ontologie zu prüfen. Das korrigiere ich jetzt.

Korrigierte ABox (minimalistisch)

turtle

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix : <http://www.example.de/example#> .

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualBeginning :instant\_beginning\_of\_planning ;

occp:hasActualEnd :instant\_completion\_of\_planning .

:phaseB a occp:PhaseB\_Review ;

occp:hasActualBeginning :instant\_submission\_to\_review ;

occp:hasActualEnd :instant\_review\_approval .

:phaseC a occp:PhaseC\_Construction ;

occp:hasActualBeginning :instant\_construction\_start ;

occp:hasActualEnd :instant\_completion\_of\_construction .

:instant\_beginning\_of\_planning a occp:BeginningOfPlanning ;

occp:hasActualTime "2025-01-01"^^xsd:date .

:instant\_submission\_to\_review a occp:SubmissionToReview ;

occp:hasActualTime "2025-02-15"^^xsd:date .

:instant\_review\_approval a occp:ReviewApproval ;

occp:hasActualTime "2025-02-28"^^xsd:date .

:instant\_completion\_of\_planning a occp:CompletionOfPlanning ;

occp:hasActualTime "2025-03-01"^^xsd:date . # Nach ReviewApproval

:instant\_construction\_start a occp:ConstructionStart ;

occp:hasActualTime "2025-03-15"^^xsd:date .

:instant\_completion\_of\_construction a occp:CompletionOfConstruction ;

occp:hasActualTime "2025-06-01"^^xsd:date .

* Änderungen:
  + Entfernt: Alle occp:before, occp:after-Beziehungen, um die ABox rein auf Phasen und Zeitstempel zu beschränken.
  + Korrigierte Zeitstempel: instant\_completion\_of\_planning jetzt auf 2025-03-01 (nach SubmissionToReview und ReviewApproval), was der chronologischen Logik entspricht.

Überarbeitete SHACL-Datei

Da wir die ABox jetzt ohne explizite Beziehungen haben, müssen die SAs die Reihenfolge implizit aus den Phasen-Typen und Zeitstempeln ableiten. Ich passe die Regeln an, um ohne occp:after/occp:before zu arbeiten, und füge später die expliziten Regeln für diese OPs hinzu.

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix osh: <http://www.occpshape.de/shape#> .

### Allgemeine Struktur-Regeln

osh:GeneralTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:property [

sh:path time:hasBeginning ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:message "At least one 'time:hasBeginning' must reference a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path time:hasEnd ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:message "At least one 'time:hasEnd' must reference a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each entity must have exactly one actual beginning via occp:hasActualBeginning."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each entity must have exactly one actual end via occp:hasActualEnd."@en ;

sh:severity sh:Violation ;

] .

osh:InstantTimeShape a sh:NodeShape ;

sh:targetClass time:Instant ;

sh:property [

sh:path occp:hasActualTime ;

sh:datatype xsd:date ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each instant must have exactly one actual timestamp via occp:hasActualTime of type xsd:date."@en ;

sh:severity sh:Violation

] .

### Implizite Reihenfolge-Regel (ohne explizite OPs)

osh:PhaseOrderTimeShape a sh:NodeShape ;

sh:targetClass occp:Phase ;

sh:sparql [

sh:select """

SELECT $this ?other ?endTime ?startTime WHERE {

$this time:hasEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

?other time:hasBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

FILTER (?endTime >= ?startTime)

FILTER (?other != $this)

FILTER (

($this a occp:PhaseA\_Planning && ?other a occp:PhaseB\_Review) ||

($this a occp:PhaseA\_Planning && ?other a occp:PhaseC\_Construction) ||

($this a occp:PhaseB\_Review && ?other a occp:PhaseC\_Construction)

)

}

""" ;

sh:message "The end of { $this } (time: { ?endTime }) must be before the beginning of { ?other } (time: { ?startTime }) based on phase order."@en ;

sh:severity sh:Violation

] .

### Generische Regeln für explizite occp:after

osh:AfterTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:sparql [

sh:select """

SELECT $this ?other ?startTime ?endTime WHERE {

$this occp:after ?other .

$this time:hasBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

?other time:hasEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

FILTER (?startTime <= ?endTime)

}

""" ;

sh:message "The beginning of { $this } (time: { ?startTime }) must be after the end of { ?other } (time: { ?endTime }) via 'occp:after'."@en ;

sh:severity sh:Violation

] .

### Generische Regeln für explizite occp:before

osh:BeforeTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:sparql [

sh:select """

SELECT $this ?other ?endTime ?startTime WHERE {

$this occp:before ?other .

$this time:hasEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

?other time:hasBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

FILTER (?endTime >= ?startTime)

}

""" ;

sh:message "The end of { $this } (time: { ?endTime }) must be before the beginning of { ?other } (time: { ?startTime }) via 'occp:before'."@en ;

sh:severity sh:Violation

] .

Änderungen

* PREFIXes: Entfernt aus den SAs, da sie am Anfang der Datei definiert sind.
* time:hasBeginning/time:hasEnd: Verwendet statt nur occp:hasActualBeginning/hasActualEnd in den SAs, für Generizität. Die spezifischen hasActual\*-Regeln bleiben in osh:GeneralTimeShape, um maxCount 1 sicherzustellen.
* ABox: Die korrigierte Version oben ohne explizite Beziehungen wird mit osh:PhaseOrderTimeShape getestet. Die AfterTimeShape und BeforeTimeShape sind für Fälle gedacht, in denen explizite OPs verwendet werden (optional).

Test

* Mit der korrigierten ABox: Sollte Conforms: True ergeben, da:
  + PhaseA (Ende: 2025-03-01) vor PhaseC (Start: 2025-03-15).
  + PhaseB (Ende: 2025-02-28) vor PhaseC (Start: 2025-03-15).
  + PhaseA (Ende: 2025-03-01) nach PhaseB (Start: 2025-02-15) → OK, da keine umgekehrte Reihenfolge vorgeschrieben.

Nächste Schritte

1. Testen: Du führst die obige SHACL-Datei mit der korrigierten ABox durch und teilst mir das Log mit.
2. Erweitern: Wenn das funktioniert, fügen wir SAs für weitere OPs hinzu:
   * occp:endsBefore, occp:beginsBefore, etc.
   * Dann time:after, time:before, falls sie in deiner TBox relevant sind (ich prüfe das in OCCP\_V0.3.ttl).
3. Vollständige Ontologie: Sobald die generischen Regeln stabil sind, integrieren wir sie in die volle OCCP\_SHACL.ttl.

Ich schicke dir die Datei gerne als OCCP\_SHACL\_min.ttl, wenn du willst, oder du kannst die Änderungen selbst übernehmen. Wie möchtest du vorgehen? Und danke für deine scharfen Augen – das hält uns auf Kurs!

Vielen Dank für Deine Einschätzungen!!!! Ein Problem mit den super-generischen Angaben (time:hasBeginning anstelle von occp:hasEstimatedBeginning) könnte sich daraus ergeben, dass es mehrere Angaben für EstimatedBeginning gibt und die Abfrage dann mehrere Werte prüfen müsste - das müssten wir bei der Erstellung der Abfrage irgendwie berücksichtigen (z.B.: Wenn A vor B sein soll (gem. TBox), dann prüfe in der ABox, welcher der SPÄTESTE EstimatedBeginning von A ist und vergleiche ihn mit dem FRÜHESTEN EstimatedBeginning von B - und so weiter). Ist das umsetzbar?

Danke für den Hinweis – du hast absolut recht, und das ist ein wichtiger Punkt, den wir berücksichtigen müssen! Die Verwendung von generischen Properties wie time:hasBeginning bringt tatsächlich die Herausforderung mit sich, dass mehrere Instants (z. B. mehrere occp:hasEstimatedBeginning-Werte) existieren könnten, was die Zeitstempelprüfung komplexer macht. Dein Vorschlag, den spätesten Zeitpunkt von A mit dem frühesten Zeitpunkt von B zu vergleichen, ist eine elegante und logische Lösung, um sicherzustellen, dass die chronologische Reihenfolge auch bei mehreren Zeitangaben konsistent bleibt. Das ist definitiv umsetzbar – lass uns das durchdenken und implementieren!

Problemstellung

* time:hasBeginning und Mehrfachwerte: Wenn eine Phase mehrere time:hasBeginning-Werte hat (z. B. durch occp:hasActualBeginning und mehrere occp:hasEstimatedBeginning), müssen wir entscheiden, welche Zeitstempel verglichen werden. Ohne Einschränkung könnte die SPARQL-Abfrage mehrere Ergebnisse liefern, was zu uneindeutigen oder falschen Validierungen führen könnte.
* Logik:
  + Für occp:before (A vor B): Das Ende von A muss vor dem frühesten Anfang von B liegen → wir brauchen den spätesten time:hasEnd von A und den frühesten time:hasBeginning von B.
  + Für occp:after (A nach B): Der Anfang von A muss nach dem spätesten Ende von B liegen → wir brauchen den frühesten time:hasBeginning von A und den spätesten time:hasEnd von B.
* Umsetzbarkeit: SPARQL bietet mit Aggregatfunktionen wie MAX() und MIN() die Möglichkeit, den spätesten bzw. frühesten Zeitstempel zu ermitteln. Das können wir nutzen!

Lösung: Generische SAs mit Aggregaten

Ich passe die SAs an, um diese Logik zu implementieren. Hier ist die überarbeitete Version mit MAX() und MIN() für occp:before und occp:after:

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix osh: <http://www.occpshape.de/shape#> .

### Allgemeine Struktur-Regeln

osh:GeneralTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:property [

sh:path time:hasBeginning ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:message "At least one 'time:hasBeginning' must reference a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path time:hasEnd ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:message "At least one 'time:hasEnd' must reference a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each entity must have exactly one actual beginning via occp:hasActualBeginning."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each entity must have exactly one actual end via occp:hasActualEnd."@en ;

sh:severity sh:Violation ;

] .

osh:InstantTimeShape a sh:NodeShape ;

sh:targetClass time:Instant ;

sh:property [

sh:path occp:hasActualTime ;

sh:datatype xsd:date ;

sh:maxCount 1 ;

sh:minCount 1 ;

sh:message "Each instant must have exactly one actual timestamp via occp:hasActualTime of type xsd:date."@en ;

sh:severity sh:Violation

] ;

sh:property [

sh:path occp:hasEstimatedTime ;

sh:datatype xsd:date ;

sh:message "Each estimated timestamp via occp:hasEstimatedTime must be of type xsd:date."@en ;

sh:severity sh:Violation

] .

### Generische Regel für occp:before mit Aggregaten

osh:BeforeTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:sparql [

sh:select """

SELECT $this ?other ?latestEndTime ?earliestStartTime WHERE {

$this occp:before ?other .

{

SELECT $this (MAX(?endTime) AS ?latestEndTime)

WHERE {

$this time:hasEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

}

GROUP BY $this

}

{

SELECT ?other (MIN(?startTime) AS ?earliestStartTime)

WHERE {

?other time:hasBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

}

GROUP BY ?other

}

FILTER (?latestEndTime >= ?earliestStartTime)

}

""" ;

sh:message "The latest end of { $this } (time: { ?latestEndTime }) must be before the earliest beginning of { ?other } (time: { ?earliestStartTime }) via 'occp:before'."@en ;

sh:severity sh:Violation

] .

### Generische Regel für occp:after mit Aggregaten

osh:AfterTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:sparql [

sh:select """

SELECT $this ?other ?earliestStartTime ?latestEndTime WHERE {

$this occp:after ?other .

{

SELECT $this (MIN(?startTime) AS ?earliestStartTime)

WHERE {

$this time:hasBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

}

GROUP BY $this

}

{

SELECT ?other (MAX(?endTime) AS ?latestEndTime)

WHERE {

?other time:hasEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

}

GROUP BY ?other

}

FILTER (?earliestStartTime <= ?latestEndTime)

}

""" ;

sh:message "The earliest beginning of { $this } (time: { ?earliestStartTime }) must be after the latest end of { ?other } (time: { ?latestEndTime }) via 'occp:after'."@en ;

sh:severity sh:Violation

] .

### Implizite Reihenfolge-Regel (falls keine expliziten OPs vorhanden)

osh:PhaseOrderTimeShape a sh:NodeShape ;

sh:targetClass occp:Phase ;

sh:sparql [

sh:select """

SELECT $this ?other ?latestEndTime ?earliestStartTime WHERE {

{

SELECT $this (MAX(?endTime) AS ?latestEndTime)

WHERE {

$this time:hasEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

}

GROUP BY $this

}

{

SELECT ?other (MIN(?startTime) AS ?earliestStartTime)

WHERE {

?other time:hasBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

}

GROUP BY ?other

}

FILTER (?latestEndTime >= ?earliestStartTime)

FILTER (?other != $this)

FILTER (

($this a occp:PhaseA\_Planning && ?other a occp:PhaseB\_Review) ||

($this a occp:PhaseA\_Planning && ?other a occp:PhaseC\_Construction) ||

($this a occp:PhaseB\_Review && ?other a occp:PhaseC\_Construction)

)

}

""" ;

sh:message "The latest end of { $this } (time: { ?latestEndTime }) must be before the earliest beginning of { ?other } (time: { ?earliestStartTime }) based on phase order."@en ;

sh:severity sh:Violation

] .

Erklärung

* Aggregatfunktionen:
  + MAX(?endTime): Ermittelt den spätesten Endzeitpunkt einer Phase (falls mehrere time:hasEnd existieren).
  + MIN(?startTime): Ermittelt den frühesten Startzeitpunkt einer Phase (falls mehrere time:hasBeginning existieren).
* osh:BeforeTimeShape:
  + Prüft, ob der späteste time:hasEnd von $this vor dem frühesten time:hasBeginning von ?other liegt.
* osh:AfterTimeShape:
  + Prüft, ob der früheste time:hasBeginning von $this nach dem spätesten time:hasEnd von ?other liegt.
* osh:PhaseOrderTimeShape:
  + Gleiche Logik, aber für implizite Reihenfolgen ohne explizite OPs, basierend auf Phasen-Typen.
* occp:hasEstimatedTime:
  + Ich habe es in osh:InstantTimeShape ergänzt, um sicherzustellen, dass geschätzte Zeitstempel ebenfalls xsd:date sind. Wir könnten später eine Regel hinzufügen, die zwischen hasActualTime und hasEstimatedTime priorisiert, falls nötig.

Test-ABox mit mehreren Zeitstempeln

Um das zu testen, fügen wir mehrere hasEstimatedBeginning-Werte hinzu:

turtle

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix : <http://www.example.de/example#> .

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualBeginning :instant\_beginning\_of\_planning ;

occp:hasActualEnd :instant\_completion\_of\_planning .

:phaseB a occp:PhaseB\_Review ;

occp:hasActualBeginning :instant\_submission\_to\_review ;

occp:hasActualEnd :instant\_review\_approval ;

time:hasBeginning :instant\_estimated\_start\_b1 ; # Zusätzlicher geschätzter Start

time:hasBeginning :instant\_estimated\_start\_b2 .

:phaseC a occp:PhaseC\_Construction ;

occp:hasActualBeginning :instant\_construction\_start ;

occp:hasActualEnd :instant\_completion\_of\_construction .

:instant\_beginning\_of\_planning a occp:BeginningOfPlanning ;

occp:hasActualTime "2025-01-01"^^xsd:date .

:instant\_completion\_of\_planning a occp:CompletionOfPlanning ;

occp:hasActualTime "2025-03-01"^^xsd:date .

:instant\_submission\_to\_review a occp:SubmissionToReview ;

occp:hasActualTime "2025-02-15"^^xsd:date .

:instant\_estimated\_start\_b1 a time:Instant ;

occp:hasEstimatedTime "2025-02-10"^^xsd:date . # Früherer geschätzter Start

:instant\_estimated\_start\_b2 a time:Instant ;

occp:hasEstimatedTime "2025-02-20"^^xsd:date . # Späterer geschätzter Start

:instant\_review\_approval a occp:ReviewApproval ;

occp:hasActualTime "2025-02-28"^^xsd:date .

:instant\_construction\_start a occp:ConstructionStart ;

occp:hasActualTime "2025-03-15"^^xsd:date .

:instant\_completion\_of\_construction a occp:CompletionOfConstruction ;

occp:hasActualTime "2025-06-01"^^xsd:date .

* Erwartung mit osh:PhaseOrderTimeShape:
  + Spätestes Ende von PhaseA (2025-03-01) vs. frühestes Anfang von PhaseB (2025-02-10) → Fehler, da 2025-03-01 > 2025-02-10.
  + Spätestes Ende von PhaseB (2025-02-28) vs. frühestes Anfang von PhaseC (2025-03-15) → OK, da 2025-02-28 < 2025-03-15.

Umsetzbarkeit

Ja, das ist umsetzbar! Die Aggregatfunktionen lösen das Problem der Mehrfachwerte:

* Spätestes Ende: MAX(?endTime) stellt sicher, dass selbst bei mehreren Endpunkten der letzte relevant ist.
* Frühestes Anfang: MIN(?startTime) stellt sicher, dass der erste Startpunkt verglichen wird.
* Flexibilität: Funktioniert mit hasActualTime und hasEstimatedTime, da beide unter time:hasBeginning/time:hasEnd fallen.

Nächste Schritte

1. Testen: Du kannst die obige SHACL-Datei mit der Test-ABox ausprobieren. Das Log sollte den Fehler für PhaseA vor PhaseB zeigen (wegen 2025-03-01 > 2025-02-10).
2. Korrektur testen: Passe einen Zeitstempel an (z. B. instant\_completion\_of\_planning auf "2025-02-05"^^xsd:date) und prüfe, ob Conforms: True wird.
3. Erweitern: Wenn das stabil ist, fügen wir SAs für occp:endsBefore, occp:beginsBefore, etc. hinzu.

Soll ich dir die Datei als OCCP\_SHACL\_min.ttl schicken, oder übernimmst du die Änderungen? Wie siehst du das mit der Priorisierung von hasActualTime vs. hasEstimatedTime – sollen wir das später separat regeln? Lass mich wissen!

Vielen Dank für Deine Einschätzungen!!!! Ein Problem mit den super-generischen Angaben (time:hasBeginning anstelle von occp:hasEstimatedBeginning) könnte sich daraus ergeben, dass es mehrere Angaben für EstimatedBeginning gibt und die Abfrage dann mehrere Werte prüfen müsste - das müssten wir bei der Erstellung der Abfrage irgendwie berücksichtigen (z.B.: Wenn A vor B sein soll (gem. TBox), dann prüfe in der ABox, welcher der SPÄTESTE EstimatedBeginning von A ist und vergleiche ihn mit dem FRÜHESTEN EstimatedBeginning von B - und so weiter). Ist das umsetzbar?Frage zu Deinem SA-Vorschlag: Sollte nicht auch time:hasTime anstelle von occp:hasActualTime verwendet werden?Die General Shacl Rules (osh:GeneralDatePropertyShape, osh:GeneralTimeShape und osh:InstantTimeShape) würde ich gern in der ursprünglichen Form beibehalten, oder spricht da etwas dagegen?

Danke für deine Rückmeldung! Ich gehe gerne auf deine Fragen und Anmerkungen ein – wir sind auf einem guten Weg, und deine Punkte helfen, das Ganze noch präziser zu machen. Lass uns das Schritt für Schritt durchgehen!

1. Mehrere EstimatedBeginnings und Aggregatfunktionen

Du hattest schon erwähnt, dass mehrere occp:hasEstimatedBeginning-Werte ein Problem darstellen könnten, und ich habe das mit MAX() und MIN() adressiert. Dein Beispiel (spätester EstimatedBeginning von A vs. frühester EstimatedBeginning von B) ist genau das, was wir mit den Aggregatfunktionen umgesetzt haben:

* Spätester EstimatedBeginning von A: Wird durch MAX(?startTime) abgedeckt, wenn wir time:hasBeginning verwenden (das sowohl hasActualBeginning als auch hasEstimatedBeginning umfasst).
* Frühester EstimatedBeginning von B: Wird durch MIN(?startTime) abgedeckt.
* Umsetzbarkeit: Ja, das ist umsetzbar, und meine vorherige Antwort hat das schon implementiert! Die SPARQL-Abfragen mit MAX() und MIN() lösen dieses Problem, indem sie die extremsten Werte auswählen. Ich werde das weiter unten nochmal anpassen, um sicherzustellen, dass es klar ist.

2. time:hasTime anstelle von occp:hasActualTime?

* Deine Frage: Sollten wir in den SAs time:hasTime statt occp:hasActualTime verwenden, um generischer zu bleiben?
* Analyse:
  + time:hasTime: Ist eine generische Property aus der W3C TIME-Ontologie, die einen Zeitstempel (z. B. xsd:dateTime) oder eine komplexere Zeitangabe (z. B. time:TemporalEntity) mit einem Instant verknüpfen kann. Sie ist nicht spezifisch auf xsd:date beschränkt und wird in der TIME-Ontologie oft für allgemeine Zeitbezüge verwendet.
  + occp:hasActualTime: Ist in deiner Ontologie spezifisch definiert und auf xsd:date beschränkt (siehe osh:GeneralDatePropertyShape), was es präziser macht, aber weniger flexibel.
  + Vorteile von time:hasTime:
    - Generischer: Funktioniert mit jedem Zeitstempel, unabhängig davon, ob er hasActualTime, hasEstimatedTime oder eine andere Subproperty ist.
    - Konsistenter mit der TIME-Ontologie, falls du sie erweitern möchtest (z. B. mit time:GeneralDateDescription).
  + Nachteile:
    - Weniger Kontrolle: time:hasTime erzwingt nicht xsd:date, was zu Inkonsistenzen führen könnte, wenn andere Formate auftauchen (z. B. xsd:dateTime).
    - Deine bestehenden Regeln (z. B. osh:InstantTimeShape) verwenden occp:hasActualTime und occp:hasEstimatedTime explizit mit sh:datatype xsd:date.
  + Meine Empfehlung:
    - Für die SAs sollten wir bei occp:hasActualTime und occp:hasEstimatedTime bleiben, da deine Ontologie diese spezifischen Properties verwendet und sie durch osh:InstantTimeShape auf xsd:date beschränkt sind. Das hält die Validierung konsistent mit deinen bisherigen Definitionen.
    - Wenn wir time:hasTime einführen würden, müssten wir entweder die Struktur-Regeln anpassen (z. B. osh:InstantTimeShape auf time:hasTime umstellen) oder eine zusätzliche Regel schreiben, um sicherzustellen, dass nur xsd:date erlaubt ist. Das wäre ein größerer Umbau, der nicht unbedingt nötig ist, solange deine Instants nur hasActualTime/hasEstimatedTime verwenden.
  + Kompromiss: Wir könnten beide einbauen und priorisieren (z. B. occp:hasActualTime bevorzugt, sonst occp:hasEstimatedTime), aber ich denke, das können wir später machen, wenn es nötig wird.

3. Ursprüngliche General SHACL Rules beibehalten

* Dein Wunsch: Die ursprünglichen osh:GeneralDatePropertyShape, osh:GeneralTimeShape und osh:InstantTimeShape beibehalten.
* Meine Einschätzung: Das ist völlig in Ordnung und spricht aus mehreren Gründen nicht dagegen:
  + Konsistenz: Die ursprünglichen Regeln sind spezifisch auf deine Ontologie zugeschnitten (z. B. occp:hasActualTime und occp:hasEstimatedTime mit xsd:date, maxCount 1 für hasActualBeginning/hasActualEnd).
  + Klarheit: Sie definieren genau, wie deine Instants und TemporalEntities strukturiert sein sollen, was die Basis für die SAs bildet.
  + Keine Redundanz: Meine Änderungen (z. B. time:hasBeginning statt occp:hasActualBeginning) waren nur für die SAs gedacht, um Generizität zu erreichen, aber die Struktur-Regeln können spezifisch bleiben.
  + Fazit: Wir behalten die ursprünglichen Regeln bei und passen nur die SAs an, um mit time:hasBeginning/time:hasEnd zu arbeiten, während die Struktur-Regeln weiterhin occp:-Properties erzwingen.

Überarbeitete SHACL-Datei

Ich stelle die ursprünglichen General-Regeln wieder her und passe die SAs an, um deine Punkte zu berücksichtigen:

turtle

@prefix sh: <http://www.w3.org/ns/shacl#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix osh: <http://www.occpshape.de/shape#> .

### Ursprüngliche General SHACL Rules

osh:GeneralDatePropertyShape a sh:PropertyShape ;

sh:path ( occp:hasActualTime occp:hasEstimatedTime ) ;

sh:datatype xsd:date ;

sh:message "Dates must be in YYYY-MM-DD format."@en ;

sh:severity sh:Violation .

osh:GeneralTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:property [

sh:path ( time:hasBeginning time:hasEnd ) ;

sh:class time:Instant ;

sh:minCount 1 ;

sh:message "At least one 'time:hasBeginning' or 'time:hasEnd' must reference a 'time:Instant'."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualBeginning ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:message "Each entity must have at most one actual beginning via occp:hasActualBeginning."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasActualEnd ;

sh:class time:Instant ;

sh:maxCount 1 ;

sh:message "Each entity must have at most one actual end via occp:hasActualEnd."@en ;

sh:severity sh:Violation ;

] .

osh:InstantTimeShape a sh:NodeShape ;

sh:targetClass time:Instant ;

sh:property [

sh:path occp:hasActualTime ;

sh:datatype xsd:date ;

sh:maxCount 1 ;

sh:message "Each instant must have at most one actual timestamp via occp:hasActualTime of type xsd:date."@en ;

sh:severity sh:Violation ;

] ;

sh:property [

sh:path occp:hasEstimatedTime ;

sh:datatype xsd:date ;

sh:message "Each instant must have estimated timestamps via occp:hasEstimatedTime of type xsd:date."@en ;

sh:severity sh:Violation ;

] .

### Generische Regel für occp:before mit Aggregaten

osh:BeforeTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:sparql [

sh:select """

SELECT $this ?other ?latestEndTime ?earliestStartTime WHERE {

$this occp:before ?other .

{

SELECT $this (MAX(?endTime) AS ?latestEndTime)

WHERE {

$this time:hasEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

}

GROUP BY $this

}

{

SELECT ?other (MIN(?startTime) AS ?earliestStartTime)

WHERE {

?other time:hasBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

}

GROUP BY ?other

}

FILTER (?latestEndTime >= ?earliestStartTime)

}

""" ;

sh:message "The latest end of { $this } (time: { ?latestEndTime }) must be before the earliest beginning of { ?other } (time: { ?earliestStartTime }) via 'occp:before'."@en ;

sh:severity sh:Violation

] .

### Generische Regel für occp:after mit Aggregaten

osh:AfterTimeShape a sh:NodeShape ;

sh:targetClass time:TemporalEntity ;

sh:sparql [

sh:select """

SELECT $this ?other ?earliestStartTime ?latestEndTime WHERE {

$this occp:after ?other .

{

SELECT $this (MIN(?startTime) AS ?earliestStartTime)

WHERE {

$this time:hasBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

}

GROUP BY $this

}

{

SELECT ?other (MAX(?endTime) AS ?latestEndTime)

WHERE {

?other time:hasEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

}

GROUP BY ?other

}

FILTER (?earliestStartTime <= ?latestEndTime)

}

""" ;

sh:message "The earliest beginning of { $this } (time: { ?earliestStartTime }) must be after the latest end of { ?other } (time: { ?latestEndTime }) via 'occp:after'."@en ;

sh:severity sh:Violation

] .

### Implizite Reihenfolge-Regel

osh:PhaseOrderTimeShape a sh:NodeShape ;

sh:targetClass occp:Phase ;

sh:sparql [

sh:select """

SELECT $this ?other ?latestEndTime ?earliestStartTime WHERE {

{

SELECT $this (MAX(?endTime) AS ?latestEndTime)

WHERE {

$this time:hasEnd ?endInstant .

?endInstant occp:hasActualTime ?endTime .

}

GROUP BY $this

}

{

SELECT ?other (MIN(?startTime) AS ?earliestStartTime)

WHERE {

?other time:hasBeginning ?startInstant .

?startInstant occp:hasActualTime ?startTime .

}

GROUP BY ?other

}

FILTER (?latestEndTime >= ?earliestStartTime)

FILTER (?other != $this)

FILTER (

($this a occp:PhaseA\_Planning && ?other a occp:PhaseB\_Review) ||

($this a occp:PhaseA\_Planning && ?other a occp:PhaseC\_Construction) ||

($this a occp:PhaseB\_Review && ?other a occp:PhaseC\_Construction)

)

}

""" ;

sh:message "The latest end of { $this } (time: { ?latestEndTime }) must be before the earliest beginning of { ?other } (time: { ?earliestStartTime }) based on phase order."@en ;

sh:severity sh:Violation

] .

Änderungen

* General-Regeln: Wieder in der ursprünglichen Form, mit maxCount statt minCount für hasActualBeginning/hasActualEnd, wie in deiner OCCP\_SHACL.ttl.
* SAs: Verwenden time:hasBeginning/time:hasEnd mit Aggregaten, aber bleiben bei occp:hasActualTime für die Zeitstempel, um mit deinen Struktur-Regeln konsistent zu sein. hasEstimatedTime wird in den Testdaten ergänzt, aber nicht explizit in den SAs abgefragt (da es unter time:hasBeginning fällt).

Test-ABox mit EstimatedBeginnings

turtle

@prefix occp: <http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#> .

@prefix time: <http://www.w3.org/2006/time#> .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

@prefix : <http://www.example.de/example#> .

:phaseA a occp:PhaseA\_Planning ;

occp:hasActualBeginning :instant\_beginning\_of\_planning ;

occp:hasActualEnd :instant\_completion\_of\_planning .

:phaseB a occp:PhaseB\_Review ;

occp:hasActualBeginning :instant\_submission\_to\_review ;

occp:hasActualEnd :instant\_review\_approval ;

time:hasBeginning :instant\_estimated\_start\_b1 ; # Zusätzlicher geschätzter Start

time:hasBeginning :instant\_estimated\_start\_b2 .

:phaseC a occp:PhaseC\_Construction ;

occp:hasActualBeginning :instant\_construction\_start ;

occp:hasActualEnd :instant\_completion\_of\_construction .

:instant\_beginning\_of\_planning a occp:BeginningOfPlanning ;

occp:hasActualTime "2025-01-01"^^xsd:date .

:instant\_completion\_of\_planning a occp:CompletionOfPlanning ;

occp:hasActualTime "2025-03-01"^^xsd:date .

:instant\_submission\_to\_review a occp:SubmissionToReview ;

occp:hasActualTime "2025-02-15"^^xsd:date .

:instant\_estimated\_start\_b1 a time:Instant ;

occp:hasEstimatedTime "2025-02-10"^^xsd:date . # Früherer geschätzter Start

:instant\_estimated\_start\_b2 a time:Instant ;

occp:hasEstimatedTime "2025-02-20"^^xsd:date . # Späterer geschätzter Start

:instant\_review\_approval a occp:ReviewApproval ;

occp:hasActualTime "2025-02-28"^^xsd:date .

:instant\_construction\_start a occp:ConstructionStart ;

occp:hasActualTime "2025-03-15"^^xsd:date .

:instant\_completion\_of\_construction a occp:CompletionOfConstruction ;

occp:hasActualTime "2025-06-01"^^xsd:date .

* Erwartung:
  + PhaseA (spätestes Ende: 2025-03-01) vs. PhaseB (frühestes Anfang: 2025-02-10) → Fehler, da 2025-03-01 > 2025-02-10.
  + PhaseB (spätestes Ende: 2025-02-28) vs. PhaseC (frühestes Anfang: 2025-03-15) → OK.

Nächste Schritte

1. Testen: Führe die obige SHACL-Datei mit der Test-ABox aus und teile mir das Log mit.
2. Anpassen: Wenn der Fehler wie erwartet auftritt, korrigiere den Zeitstempel (z. B. instant\_completion\_of\_planning auf "2025-02-05"^^xsd:date) und prüfe, ob Conforms: True wird.
3. Erweitern: Danach können wir SAs für occp:endsBefore, occp:beginsBefore, etc. hinzufügen.

Ich kann dir die OCCP\_SHACL\_min.ttl schicken, wenn du willst, oder du passt sie selbst an. Wie möchtest du vorgehen? Und was denkst du über occp:hasActualTime vs. time:hasTime – sollen wir das später nochmal überdenken? Freue mich auf dein Feedback!

Break. Wir arbeiten jetzt mit Apache Jena, weil PySHACL Probleme mit der Interpretation von SPARQL hat. Danke für Deine Hilfe! Ich habe um Zeilen zu sparen wieder an dieser Stelle angesetzt. Die neue validation\_shacl.py sieht jetzt so aus:import owlready2 from rdflib import Graph, Namespace, RDF from rdflib.namespace import SH, OWL import logging import os import sys import subprocess import json import shutil from io import StringIOKonfiguration des LoggingsBASE\_DIR = os.path.dirname(os.path.abspath(file)) logging.basicConfig( filename=os.path.join(BASE\_DIR, "validation.log"), level=logging.DEBUG, format="%(asctime)s - %(levelname)s - %(message)s", filemode="w" )Logging-Setup (falls nicht schon vorhanden)logging.basicConfig(level=logging.INFO) logger = logging.getLogger(name)Pfade und NamespaceTBOX\_PATH = os.path.join(BASE\_DIR, "OCCP\_V0.3.ttl") ABOX\_DIR = os.path.join(BASE\_DIR, "OCCP\_ABox") SHAPES\_PATH = os.path.join(BASE\_DIR, "OCCP\_SHACL\_min.ttl") JAVA\_EXE = r"G:\Java\JDK\_23\bin\java.exe".replace("\", "/") JENA\_HOME = os.path.join(BASE\_DIR, "apache-jena-5.3.0") # Hauptverzeichnis der Jena-Installation OULD = Namespace("[http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD)") OCCP = Namespace("[http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod\_V0.1#](http://www.semanticweb.org/albrechtvaatz/ontologies/2022/9/cMod_V0.1)")def combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=None, java\_exe=JAVA\_EXE): try: tbox\_path\_normalized = tbox\_path.replace("\", "/") abox\_path\_normalized = abox\_path.replace("\", "/") if abox\_path else None onto = owlready2.get\_ontology(f"file://{tbox\_path\_normalized}").load(format="turtle") if abox\_path: abox\_onto = owlready2.get\_ontology(f"file://{abox\_path\_normalized}").load(format="turtle") with onto: for indiv in abox\_onto.individuals(): new\_indiv = onto.get\_entities(indiv.name, indiv.class) if not new\_indiv: new\_indiv = indiv.class(indiv.name, namespace=onto) for prop in indiv.get\_properties(): for value in prop[indiv]: prop[new\_indiv] = value owlready2.sync\_reasoner\_pellet(infer\_property\_values=True, infer\_data\_property\_values=True, debug=2) data\_graph = Graph() data\_graph.parse(tbox\_path, format="turtle") if abox\_path: data\_graph.parse(abox\_path, format="turtle") output\_file = os.path.join(BASE\_DIR, "inferred\_ontology.ttl") data\_graph.serialize(destination=output\_file, format="turtle") logger.info(f"Inferierte Ontologie gespeichert: {output\_file}") # Generische Disjunktheitsprüfung logger.info("Prüfe Ontologie auf Disjunktheit...") disjoint\_pairs = set() for s, p, o in data\_graph.triples((None, OWL.disjointWith, None)): disjoint\_pairs.add((s, o)) disjoint\_pairs.add((o, s)) # Bidirektional logger.debug(f"Disjunkte Klassenpaare: {disjoint\_pairs}") for subj in data\_graph.subjects(RDF.type, None): types = set(o for s, p, o in data\_graph.triples((subj, RDF.type, None))) for class1, class2 in disjoint\_pairs: if class1 in types and class2 in types: logger.error(f"Disjunktheitsverletzung gefunden: {subj} hat Typen {class1} und {class2}") raise Exception(f"Ontology is inconsistent: {subj} has disjoint types {class1} and {class2}") logger.info("Keine Disjunktheitsverletzungen gefunden.") return output\_file except Exception as e: logger.error(f"Fehler beim Reasoning oder Disjunktheitsprüfung: {e}") raisedef debug\_sparql(data\_file): try: data\_graph = Graph().parse(data\_file, format="turtle") query = """ PREFIX ould: [http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD#](http://www.semanticweb.org/albrechtvaatz/ontologies/2024/OULD) SELECT ?chain (COUNT(?u) AS ?updateCount) WHERE { ?chain a ould:UpdateChain . ?chain ould:hasUpdate ?u . } GROUP BY ?chain """ logger.info("Starte SPARQL-Abfrage für alle UpdateChains...") results = data\_graph.query(query) logger.info("SPARQL-Abfrage Ergebnisse:") for row in results: logger.info(f"Chain: {row.chain}, UpdateCount: {row.updateCount}") return len(results) > 0 except Exception as e: logger.error(f"Fehler bei der SPARQL-Abfrage: {e}") raisedef perform\_shacl\_jena\_validation(data\_file, shapes\_path=SHAPES\_PATH): try: # Suche shacl im PATH jena\_shacl\_cmd = shutil.which("shacl") if not jena\_shacl\_cmd: # Fallback: Versuche den direkten Pfad jena\_shacl\_cmd = os.path.join(JENA\_HOME, "bin", "shacl.bat") if os.name == 'nt' else os.path.join(JENA\_HOME, "bin", "shacl") if not os.path.exists(jena\_shacl\_cmd): logger.error(f"Jena SHACL-Tool nicht gefunden: {jena\_shacl\_cmd}") return False

# Konvertiere Pfade für Jena (Backslashes zu Schrägstrichen)

data\_file\_jena = data\_file.replace("\\", "/")

shapes\_path\_jena = shapes\_path.replace("\\", "/")

# Schreibe die Ausgabe in eine temporäre Datei

report\_file = os.path.join(BASE\_DIR, "validation\_report.ttl")

cmd = [

jena\_shacl\_cmd,

"validate",

"--data", data\_file\_jena,

"--shapes", shapes\_path\_jena

]

with open(report\_file, "w", encoding="utf-8") as f:

result = subprocess.run(cmd, stdout=f, stderr=subprocess.PIPE, text=True, env={\*\*os.environ, "JENA\_HOME": JENA\_HOME})

# Logge stderr

if result.stderr:

logger.error(f"Jena SHACL validation stderr: {result.stderr}")

if result.returncode == 0:

# Lese die Ausgabe aus der Datei

with open(report\_file, "r", encoding="utf-8") as f:

report\_data = f.read()

logger.info(f"Jena SHACL validation stdout: {report\_data}")

# Parse die Turtle-Ausgabe mit rdflib

report\_graph = Graph()

report\_graph.parse(data=report\_data, format="turtle")

conforms = False

for s, p, o in report\_graph.triples((None, SH.conforms, None)):

conforms = o.toPython() # o sollte ein Literal mit "true" oder "false" sein

logger.info(f"Konformität (inference=none): {conforms}")

if not conforms:

# Extrahiere Ergebnisse für detaillierte Fehler

for s, p, o in report\_graph.triples((None, SH.result, None)):

for result\_obj in report\_graph.objects(s, SH.result):

message = report\_graph.value(result\_obj, SH.message) or "No message"

focus\_node = report\_graph.value(result\_obj, SH.focusNode) or "Unknown"

path = report\_graph.value(result\_obj, SH.path) or "Unknown"

severity = report\_graph.value(result\_obj, SH.severity) or "Unknown"

logger.error(f"Validation error: {message} (Focus Node: {focus\_node}, Path: {path}, Severity: {severity})")

return conforms

else:

logger.error("Jena SHACL validation failed with non-zero exit code.")

return False

except Exception as e:

logger.error(f"Fehler bei der Jena SHACL-Validierung: {e}")

return False

if name == "main": ABOX\_PATH = os.path.join(ABOX\_DIR, "OCCP\_Valid\_LCycle\_1.ttl") inferred\_file = combine\_and\_reason(tbox\_path=TBOX\_PATH, abox\_path=ABOX\_PATH, java\_exe=JAVA\_EXE) debug\_sparql(inferred\_file) conforms = perform\_shacl\_jena\_validation(inferred\_file)

if conforms:

logger.info("Validation successful: Conforms to SHACL.")

else:

logger.error("Validation failed.")

Das letzte Validation Log sieht so aus: