A DECISION DESIGN PATTERN TO DETECT AND PRESENT THE MATURITY-LEVEL OF DECISION-RELEVANT INFORMATION

Written by:	Mario Verhaeg
Student ID number:	851210638
Thesis presentation date:	June 30, 2020
Version:	2.01, June 22, 2020

A DECISION DESIGN PATTERN TO DETECT AND PRESENT THE MATURITY-LEVEL OF DECISION-RELEVANT INFORMATION

Written by:	Mario Verhaeg
Student ID number:	851210638
Thesis presentation date:	June 30, 2020
Version:	2.01, June 22, 2020
Degree program:	Faculty Science, Open University of the Netherlands
	Computer Science Master's program
Course:	IM0004 CS Graduation Assignment Preparation
	IM990C Computer Science Graduation Assignment
Thesis supervisor:	dr. L.W. Rutledge
Second reader:	dr. Bastiaan Heeren

Acknowledgement

Studying has been a significant part of my life since I started with this journey in 2011. In the meantime, I started working at Bosch, the German industrial giant and moved to the south of Germany. This job challenges me every day and brings me to exciting parts of the world. It was quite a challenge to keep this up for almost ten years, but I made it.

My initial goal was to finish this study before my 30th birthday. Unfortunately, I was not able to achieve this goal. At the same time, the learning experience was more important for me than keeping the schedule. I have no regret from restarting several courses, including the graduation assignment, and taking my time to understand what is going on. My goal is to learn, not to impress.

The inspiration for the subject of this graduation assignment came from my work environment. I observed that we made several complex and high-impact decisions using information that was based on opinions and open for interpretation. Taking the complexity of the job, product, and market into account, I felt there should be a more structured approach to take these decisions. We, Lloyd and I, discussed the potential to use the Semantic Web to address parts of this challenge, which intrigued me.

Even though they are probably responsible for messing up my schedule, staying close to my family and friends contributed to this result. The everyday distractions they caused, the weekends travelling to the Netherlands driving 160km/h on the autobahn for six hours, the long Oktoberfest weekends, and the countless trips to the Alps kept me alive during these ten years. Lloyd kept me focused during the graduation assignment itself, and provided a continuous stream of guidance during our regular appointments, while Bastian shared his outside-in perspective. Finally, effective tooling also helps. Skype only failed me once in the last couple of months, Grammarly improved my writing skills, and Spotify shared countless hours of motivation.

Thank you for your support.

Mario

Abstract

This work proposes to use a combination of Semantic Web concepts, including consistency, inferencing, and constraints to detect premature information. We calculate the information maturity-level and present it using well-known data presentation concepts. We propose the decision design pattern to reduce the time and expertise domain experts need to build Semantic Web-based ontologies that can detect premature information and present the information maturity-level. Additionally, the transparent information maturity-level helps decision-makers to move from intuition-based decision-making to evidence-based decision-making.

Decisions, especially in the software product management discipline, are mostly complex and made based on intuition due to missing information or the lack of time to analyse the available information. A software product manager can understand if *this* is the right time to make *that* decision by understanding the information maturity-level of the decision-relevant information.

The decision design pattern calculates the information maturity-level using the completeness, reproducibility, consensus, and conflict violations. We detect the completeness, reproducibility, consensus, and conflict violations using Semantic Web constraints. Domain experts can re-use the decision design pattern and apply it for their decisions. Based on the presentation of the information maturity-level, the decision-maker can decide to continue the decision-making process or elaborate on the information.

We need to have the domain knowledge to validate the Semantic Web constraints. Therefore, we use two decisions from the software product management domain to validate our approach: requirement prioritisation and alternative solution selection. We create a supporting ontology structure for both decisions and instantiate the decision design pattern.

Key terms
Evidence-based management; EBM; generic ontology design pattern; GODP; software product management; SPM; decision making; ontology design pattern; constraints; SHACL; Semantic Web.

Contents

1 Introduction

A software product manager needs to act as a spider in a web and is involved in many disciplines, for example, product lifecycle management, product requirements engineering, release planning, road mapping, and defining the product vision (Maglyas et al., 2017). Keeping track of the sheer volume of information involved in these disciplines is a challenge. On top of that, in today's rapidly changing and uncertain environment, making strategic decisions has become increasingly complex (Büyüközkan & Feyzıoglu, 2004). The overall complexity of software product management leads to the identification of premature decision-making as one of the prime challenges in software product management (Saltan et al., 2018).

1.1 The prime challenge

Practitioners are forming communities (Mind the Product, 2019) that share experience on, for example, "Evaluating Experiments: When the Numbers lie" and "How can Enterprise Product Managers Attain Maximum Insight from Limited Datapoints?". The scientific community starts to investigate the feasibility of evidence-based decision-making in software product management, and recognises product managers need evidence-based decision-making for long-term and sustainable software product development (Saltan et al., 2018). At the same time, product managers are afraid that evidence-based decision-making reduces flexibility by formalising the decision-making process. Although the feasibility of evidence-based decision-making is still unclear, existing literature shows that the data related to, for example, sales and pricing, is available but not used for decision-making.

'We are collecting a lot of data, but simply not using it...' (Saltan et al., 2018)

Requirements prioritisation is an example that we analyse in more detail. Imagine a large multinational organisation that sells an enterprise management software product. This organisation stores information in different systems. These systems store customer-related information: the backlog management system stores the source of a requirement and the problem the customer faces, the CRM system stores the interactions with customers, and the ERP system stores revenue related information. It is very time-consuming to structure this information in a way it supports the decision-making process. Imagine this organisation sells multiple products to the same customer, and the organisation uses a different backlog management system for each product. Storing this information in various systems increases the complexity even further. At the same time, the decision-making process is complex and involves multiple stakeholders. Those stakeholders all have a different view on a situation.

Unfortunately, this study cannot fully solve this challenge. We define a general data structure to store evidence-based information and use this structure to detect premature information and present the information maturity-level. The presentation of the information maturity-level helps the decision-maker to evaluate if the quality of the information used as a basis for the decision is acceptable. The decision-maker, a human being, is eventually responsible for making the (strategic) decision.

1.2 Context

The challenge touches several disciplines in which researchers are active: decision-making, knowledge management, and for validation purposes, (software) product management. Baba and HakemZadeh, 2012 describe a theory on evidence-based decision-making that concludes that the decision-making process is not purely rational. Evidence can come from multiple (non-scientific) sources and is interpreted differently among persons. Knowledge management influences the decision-making process by, for example, the structure in which organisations store knowledge (Nicolas, 2004). The scientific community gives little attention to decision-making processes (Saltan et al., 2018). Additionally, it is not always clear to software product managers what information could serve as evidence.

1.3 Semantic web

Software product managers seem to be lacking insights into *actionable* information: information that they should use to drive decisions. The Semantic Web promises to deliver actionable information:

'The Semantic Web is a Web of actionable information...' (Shadbolt et al., 2006)

The Semantic Web transforms meaningless and unstructured information into evidence that decision-makers can use in a specific context. This study uses the Semantic Web to store decision-relevant information into an evidence-based management structure. It uses inferencing, consistency, and constraints on top of this structure to detect premature information and calculate the information maturity-level of decision-relevant information. We generalize our approach and create several ontology design patterns that domain experts can re-use to solve similar challenges in a different context.

1.4 Main research question

This work proposes to use a combination of Semantic Web concepts, including consistency, inferencing, and constraints to detect premature information. We calculate the information maturity-level and present it using well-known data presentation concepts. We propose the decision design pattern to reduce the time and expertise domain experts need to build Semantic Web-based ontologies that can detect premature information and present the information maturity-level. Additionally, the transparent information maturity-level helps decision-makers to move from intuition-based decision-making to evidence-based decision-making.

RQ: To what extent can the detection of premature information using SHACL Semantic Web constraints contribute to evidence-based decision-making?

1.5 Approach

The decision design pattern includes the evidence-based management pattern, the decision ontology pattern, and the decision presentation pattern. We create the evidence-based management pattern to prepare existing ontologies for evidence-based decision-making. The decision ontology pattern provides the structure to detect premature information. Last, the decision presentation pattern calculates the information maturity-level and presents it to the decision-maker. We use Semantic Web consistency, inferencing, and constraints. We validate the decision design pattern using two software product management decisions:

1. Scenario 1: Requirements prioritisation
2. Scenario 2: Alternative solution selection

We instantiate parts of the decision design pattern in the context of these decisions. The instantiated patterns detect if decision-relevant information is premature. We calculate and present the information maturity-level in a way the software product manager can make the decision, or elaborate further on the decision-relevant information to increase the information maturity-level.

2 Theoretical framework

The goal of the theoretical framework is to summarise the existing knowledge on the subjects related to the (main)research question. These questions define the scope of the theoretical framework:

$TF1$: How are decision-makers making decisions?

$TF2$: When is decision-relevant information premature?

$TF3$: To what extent can we detect premature information?

$TF4$: What is the impact of the presentation of the information maturity-level on a decision?

$TF5$: To what extent can we generalise our approach?

2.1 Decision making: science or art?

We typically use prior experience, intuition, or advice from others to make decisions (Harker, 1989). However, making a professional decision that might impact colleagues and customers is typically more challenging. We need to convince our stakeholders with the right arguments. Evidence-based management classifies these arguments. Briner et al., 2009 define evidence-based management as:

> *"Making decisions through the conscientious, explicit, and judicious use of four sources of information: practitioner expertise and judgment, evidence from the local context, a critical evaluation of the best available [external]research evidence, and the perspectives of those people who might be affected by the decision." (Briner et al., 2009)*

2.1.1 Decision-making model

Figure 1 presents a mixed-level model for evidence-based decision-making. This model shows that a decision-maker bases a decision on multiple evidence types. The size of the circle represents the amount of influence the evidence type has in a specific decision.

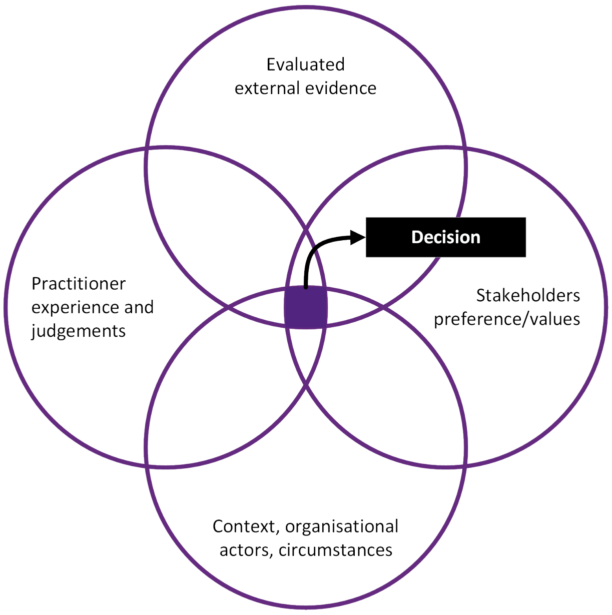

Evaluated external evidence
Decision
Practitioner experience and judgements
Stakeholders preference/values
Context, organisational actors, circumstances

Figure 1: The four elements of evidence-based management (Briner et al., 2009). The size of each circle (representing the strength of its influence) varies with each decision. For example, if we move the context, organisational actors, and circumstances further to the bottom, its influence in the decision decreases.

Evidence includes local context[1], insight from other sources, and professional experience (or *"accumulated past experience"* (Saltan et al., 2018)) (Briner et al., 2009). Different people use evidence types in different ways, depending on their personal experience. Baba and HakemZadeh, 2012 argue that evidence-based decision making is influenced by:

> *"[...] managers' preferences and values as well as stakeholders' preferences within institutional, organisational and individual contexts." (Baba & HakemZadeh, 2012).*

Evidence-based decision-making considers the way decision-makers gather the evidence, ensure this suits the usage of the evidence (methodological fit), and the context in which the information used (Baba & HakemZadeh, 2012). The quality of the information depends on its reproducibility, the transparency on evidence conflicts, and the consensus of evidence.

2.1.2 Technology-driven knowledge management

A knowledge management strategy represents how organisations have implemented knowledge management and how it impacts the firm's decision making (Nicolas, 2004). The knowledge management strategy approach structures organisational knowledge to justify strategic choices (or decisions) and can save time in the decision-making process.

2.2 Detecting premature information

We use Semantic Web technologies to detect premature information.

2.2.1 Resource description framework schema (RDFS)

Linking information starts with creating a *common understanding* of the data concepts between different contexts: this is the goal of the Resource Description Framework (RDF) (Shadbolt et al., 2006). RDF Schema (RDFS) provides data-modelling mechanisms on top of RDF, allowing it to describe groups of related resources and the relationships between those resources (Brickley et al., 2015).

2.2.2 SPARQL Query Language for RDF

The SPARQL query language can query data stored in an RDFS data model. SPARQL can be considered a data-access and filtering protocol for RDFS (Prud'Hommeaux, Seaborne, et al., 2008). SPARQL uses a table to present its output (or result set). SPARQL can be used to test data in RDFS graphs by defining constraints in SPARQL and observing the output.

SPARQL queries can select data from an ontology. If a query returns with an empty result set, the constraints described in the $WHERE$ clause of query do not match any information in the ontology. This mechanism led to SPIN (Knublauch, 2011), also known as SPARQL rules. SPIN can attach SPARQL queries to classes. The SPARQL query would define the constraints that each instance of the class needs to satisfy. Knublauch et al., 2011 submitted SPIN to the W3C. However, SPIN never made it to a recommended standard.

2.2.3 Web Ontology Language

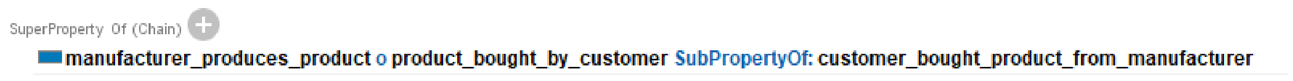
The Web Ontology Language (OWL) can be used on top of RDF(S) to increase the expressivity in the relationships between data fields. Its main goal is to *represent* knowledge (Welty et al., 2004). While RDF(S) describes the data and its relationships within a single ontology, OWL allows relationships between multiple ontologies and their data fields and supports various types of inference. Inferencing makes it easier to discover relationships between fields and reveal complexity in a structure that was not visible before (Shadbolt et al., 2006).

[1]The context for individuals is given by the organisation and the context for the organisation is given by the external environment (Johns, 2006).

The consistency on an OWL ontology can be checked by a reasoner, for example, Pellet. A reasoner includes consistency checkers based on the OWL specification (Sirin et al., 2007):

> *'An OWL consistency checker takes a document as input and returns one word being Consistent, Inconsistent, or Unknown.' (Bechhofer et al., 2004)*

Semantic web inferencing and reasoning works based on rules defined in OWL. The reasoner derives new facts from these rules. For example, the *Super Property Of (Chain)* allows defining a chain of relationships; for example, $customer\_bought\_product\_from\_manufacturer$ is a combination of two relationships: $manufacturer\_produces\_product$ and $product\_bought\_by\_customer$. When these two relationships are defined, the reasoner automatically infers the super property (figure 2). Inferencing decreases the risk of mistakes and increases the completeness of the information. As a result, inferencing decreases the risk the decision-relevant information does not meet the expected requirements.


SuperProperty Of (Chain)

manufacturer_produces_product o product_bought_by_customer SubPropertyOf: customer_bought_product_from_manufacturer

Figure 2: A super property that defines a chain of object properties. For example, the super property $customer\_bought\_product\_from\_manufacturer$ relates manufacturers with customers using the object properties $manufacturer\_produces\_product$ and $product\_bought\_by\_customer$.

2.2.4 Semantic web rule language (SWRL)

Horrocks et al., 2004 attempted to introduce data validation mechanisms in the context of the Semantic Web. SWRL extends OWL with Horn-like rules that are a form of implication between an antecedent and consequent: whenever the antecedent is true, the consequent should be true as well. SWRL never made it past its W3C submission state. Semantic Web reasoners are also able to infer new knowledge based on SWRL rules.

2.2.5 Shapes constraints language (SHACL)

OWL suffers from restrictions related to the limited possibilities for structural validation, and the built-in nature of the so-called *Open World Assumption*[2] (Boneva et al., 2017). Another limitation of OWL relates to the way how restrictions are working. For example, a person *p* can only have one father, but *p* has two individuals defined as a father. OWL assumes that these two values are representing the same real-world entity.

The W3C has accepted SHACL (Shapes Constraint Language) as a recommendation in 2017 (Knublauch & Kontokostas, 2017) to address these limitations. The main goal of SHACL is the *validation* of RDF(S) graphs against a set of conditions by defining SHACL shapes (Knublauch & Kontokostas, 2017). Figure 3 presents SHACL conceptually. SHACL can detect data quality issues (Spahiu et al., 2018) based on the definition of constraints. For example, each person needs to have precisely one last name.

[2]The open world assumption prevents a negation as failure that means that the absence of information cannot lead to any conclusion, but results in an *unknown* evaluation.

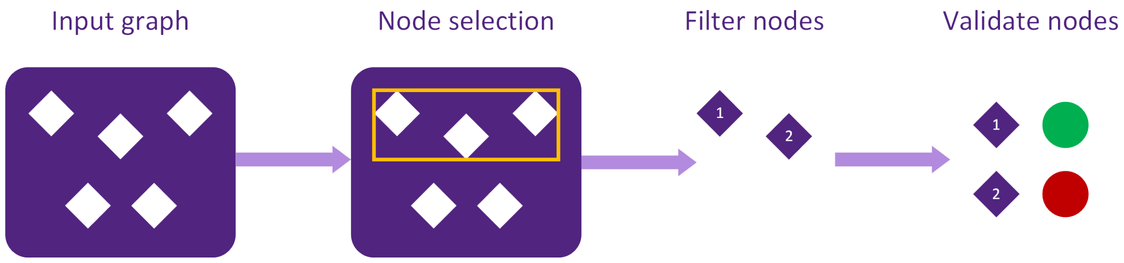


Figure 3: The main goal of SHACL is the *validation* of RDF(S) graphs against a set of conditions by defining SHACL shapes (Knublauch & Kontokostas, 2017).

2.3 Transformation of information into decisions

The way a decision support system presents information to a user has a significant impact on the quality of that system (Li et al., 2001). Decision-makers can use the information as a communication medium, a knowledge management tool, and a decision support instrument (Al-Kassab et al., 2014). We focus on the presentation of information as a decision-support instrument using graphs and charts. The presentation of information into graphs and charts enhances the capabilities of a decision-maker to process information (Coury & Boulette, 1992). However, each decision has its challenges. Selecting the wrong graph, chart, or navigation structure might lead to misleading conclusions. Therefore, the presentation needs to take the characteristics of the decision and the decision-maker into consideration (Al-Kassab et al., 2014).

There are several ways to tailor the presentation of the information to the characteristics of a decision. First, we need to select the right chart type. The available chart types include, for example, scattergraphs, line graphs, bar graphs, and pie charts (Coles, 1997, Hardin et al., 2012). A pie chart is, for example, especially useful to make it easy to understand relative proportions. A (geographical) map can make it easier to understand data that is related to multiple locations. Figure 4 presents an example in which multiple charts are combined. The combination of charts is especially useful when the information that a decision-maker uses to make a decision contains geographical information (Hardin et al., 2012).

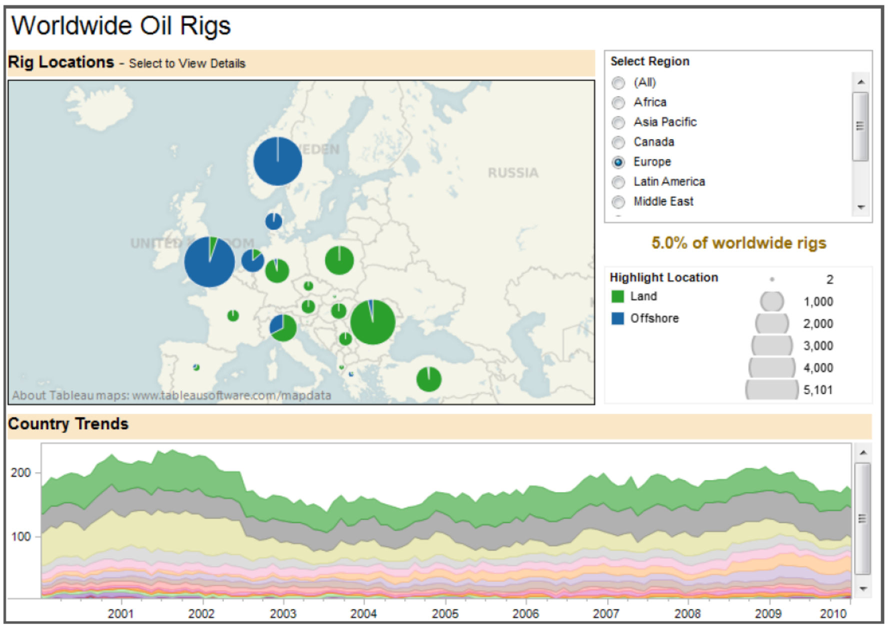


Figure 4: The worldwide oil rigs using a combination of pie charts, a geographical map, and a line chart (Hardin et al., 2012).

Once we have selected a chart, we can tailor the chart itself. We can, for example, change the way the data markers and data labels are presented (Coles, 1997). Figure 5 presents an example of a chart that is easy to interpret by the limited amount of information. It is consistently coloured, and the axis titles are explicit. Figure 6 presents an example of a chart that is more difficult to interpret by the significant difference in scales, the inconsistent colouring, and the missing definitions of the axis'.

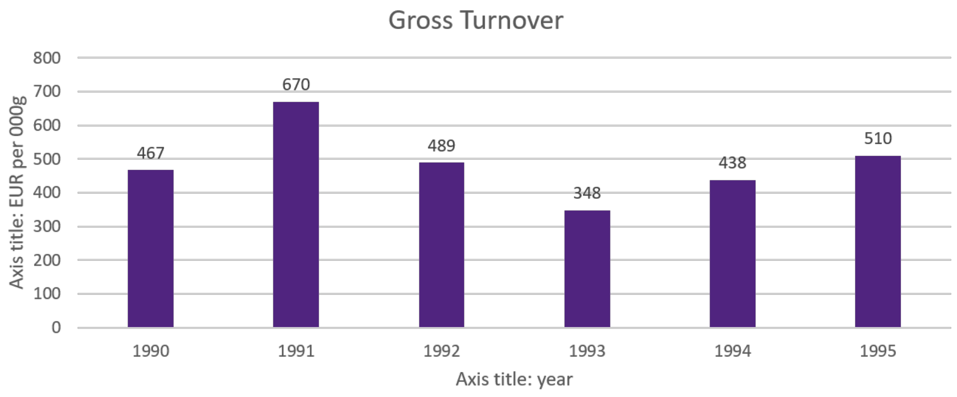


Figure 5: An example of a chart that is easy to interpret by the limited amount of information. It is consistently coloured, and the axis titles are explicit (Coles, 1997).

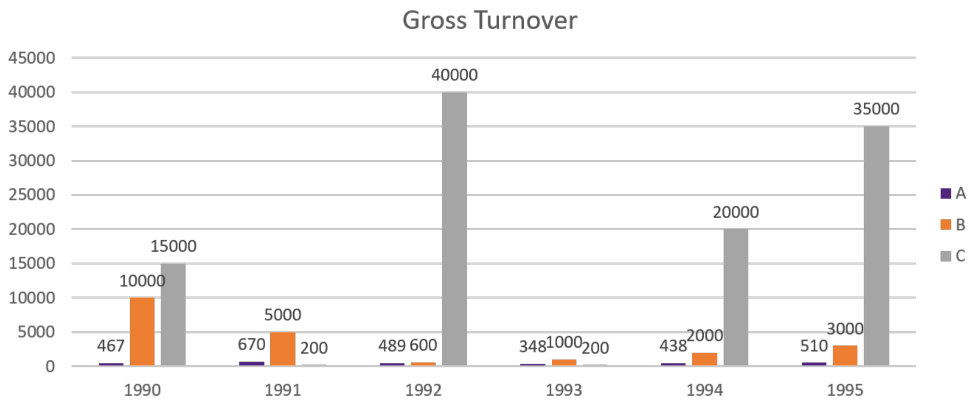


Figure 6: An example of a chart that is more difficult to interpret by the significant difference in scales, the inconsistent colouring, and the missing definitions of the axis' (Coles, 1997).

Coles, 1997 describes best practices that are useful when we want to present information with a specific goal. For example, we should limit the usage of 3D charts as they tend to obliterate data series and are challenging to interpret.

2.4 Generalisation

Kleiner, 2015 defines a design pattern as the core of a solution for similar problems. Ontologies describe concepts on a knowledge level and focus on the structure of knowledge (Devedzić, 2002). Presentation design patterns are challenging to find. However, we can interpret existing data presentation concepts like design patterns. These presentation concepts are reusable for similar problems as well.

2.4.1 Ontology design patterns

We define an ontology design pattern as an ontology configuration that effectively solves multiple problems (Soshnikov, 2003).

There are three types of stakeholders involved in ontology engineering (Krieg-Brückner et al., 2019): ontology experts, domain experts, and end-users. In general, end-users have little domain knowledge, domain experts have little ontology expertise, and ontology experts have little domain knowledge. Domain experts drive the majority of ontology development with limited involvement of ontology experts. This limited involvement might result in poor design choices (Krieg-Brückner et al., 2019). Ontology design patterns enable domain experts to re-use design decisions and best practices.

Content ontology design patterns

A content ontology design pattern includes the generic use case, specific use-case, address logic, reference ontologies, formal relationships, sensitive axioms, and the related class diagram. This approach follows software design patterns that typically also contains a problem description, suggested solution, implementation guidelines and consequences of using the pattern (Svatek, 2004).

Generic ontology design patterns

Soshnikov, 2003 introduced ontology design patterns in 2003. However, their adoption by ontology engineers has been slow (Krieg-Brückner et al., 2019). Ontology engineers have two options to use a predefined ontology design pattern:

1. Import the ontology design pattern as-is into an existing ontology. The import process might require some manual adjustments.
2. Manual redesign of the ontology design pattern, so it fits into the target ontology.

The ontology design pattern clutters the ontology with information that might not be needed. Additionally, the ontology design pattern might introduce inconsistencies into the ontology related to, for example, naming conventions. Option two is very time consuming and error-prone. Generic ontology design patterns allow ontology engineers to use ontology design patterns without cluttering the existing ontologies safely. At the same time, generic ontology design patterns prevent ontology engineers from manually redesigning the ontology to fit the ontology design pattern (Krieg-Brückner et al., 2019).

The Generic Distributed Ontology, Model and Specification Language (Generic DOL, or GDOL) is:

> *"... a meta-language that allows to define and manipulate ontologies and networks of ontologies." (Krieg-Brückner et al., 2019)*

GDOL embeds OWL expressions to, for example, extend an existing OWL ontology. For example, if A and B are two ontologies (or instantiations of generic ontology design patterns), A and B create an intersection of the two ontologies. Additionally, generic ontology design patterns can contain parameters. The parameters allow the instantiation of customised object and property names while keeping the structure of the pattern intact.

Ontology engineers can use the heterogeneous toolset (HETS) to implement the generic ontology design pattern into an ontology. The Heterogeneous Tool Set (HETS) interprets GDOL. HETS *flattens* the ontology and generates a proper OWL ontology based on the GDOL definition. For example, HETS creates a new ontology AB from the ontologies A and B.

Krieg-Brückner and Mossakowski, 2017, planned a Protégé plugin. However, there are currently no development tools available that can take care of the instantiation, extension, modification, and combination of generic ontology design patterns.

2.4.2 Information presentation design patterns

Most current work focuses on the presentation of information and aims to achieve a particular goal. A prioritisation process, for example, uses a distribution chart to visualise how stakeholders have voted for an item. In contrast, the prioritisation process uses a disagreement chart to visualise the dispersion of priorities among stakeholders (Regnell et al., 2000). However, when we take one step back, we observe that the distribution chart is a bar chart that software product managers use to compare the priority of the different items (Hardin et al., 2012). Additionally, we observe that the disagreement chart is a line chart that software product managers use to visualise the disagreement between stakeholders over the prioritised items. We consider the low-level charts as information presentation design patterns. Each chart serves as the core of a solution for similar problems.

2.5 Software product management

Product lifecycle management, product requirements engineering, release planning, road mapping and the definition of a (product) vision (Maglyas et al., 2017) represent the core activities of a software product manager. Naturally, in each of these activities, a software product manager gathers knowledge and analyses information to drive a specific decision.

Product lifecycle management manages the business processes and the related information along the entire lifecycle of the product (Matsokis & Kiritsis, 2010). An alternative way of looking at product lifecycle management focuses on knowledge management. External forces play a role in product lifecycle management as well. For example, globalisation, increased complexity, shrinkage in the product life cycle (addressing the speed of change in customer needs), and environmental issues (Ameri & Dutta, 2005).

Product requirements engineering defines problems based on (user) research by considering potential dependencies, assets, product lines and themes. The (technical) solution amends the problem description (Weerd et al., 2006). The software product manager needs to elicit and re-evaluate new requirements continuously as the market and technology evolve (Natt och Dag et al., 2005).

Release planning is considered the short-term planning process that takes care of scoping and prioritising the requirements for the next product release. Prioritising requirements can be done based on multiple data inputs, including stakeholder opinions (Weerd et al., 2006) and financial data.

Defining the product roadmap, compared to release-planning, takes care of forecasting (market trends and technology) as well as planning (products and resources) on a mid to long-term basis (Weerd et al., 2006).

The product vision is the first step in understanding why an organisation or product exists in the market. When the organisation does not understand its own business, it will start focusing on short-term issues and cost-cutting actions (Maglyas et al., 2012).

2.5.1 Concepts

Insights, opportunities, challenges, and solutions

Field research, in the context of the defined project goal, discovers problems, develops and designs solutions, tests those solutions, and confirms projects. Figure 7 presents the double diamond model and the discovery process, including field research (Mind the Product Training, 2018).

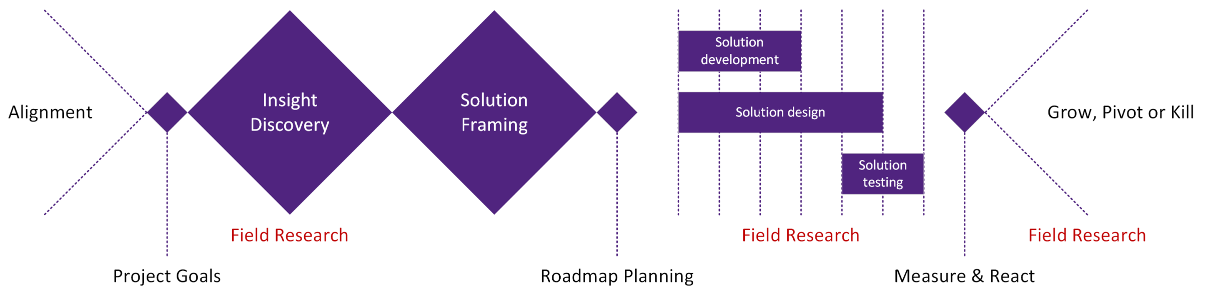


Figure 7: The double diamond model and shows how the discovery process embeds field research (Mind the Product Training, 2018).

Interesting pieces of information might surface during the field research. These pieces of information, ideally directly quoted from the source, are defined as *insights* (Mind the Product Training, 2018). An opportunity arises from a positive insight: it delights the stakeholder and motivates action. A challenge arises from a negative insight: it frustrates the stakeholder and potentially blocks the success. The product team frames a solution based on the insight. The solution goes back into the field research where a product team use new insights for further development, design, and testing. Once finished, there are three options: grow (continue), pivot (restart) or kill the project.

Goals and requirements

Requirements prioritisation decides which requirement is most important in the context of an insight. Alternative solution selection decides the best way to address an insight. Defining what a solution exactly is and where it is coming from further supports building the common understanding.

We use the definition: *'A goal is a prescriptive statement of intent the system should satisfy [...]'* (Van Lamsweerde, 2009) and define that a goal is equal to a solution. A software engineer needs a detailed view on a goal and slices the goal into multiple sub-goals or requirements: *'A requirement is a goal under the responsibility of a single agent of the software-to-be'* (Van Lamsweerde, 2009). As a result, a requirement achieves a (sub) goal. It adds customer value, even though it might not be able to address the primary goal directly. The relationship between a primary goal, potential sub-goals, and the related requirements is called a goal-model. The goal-model shows how higher-level goals are satisfied by lower-level goals (or requirements). Figure 8 presents an example of a goal-model.

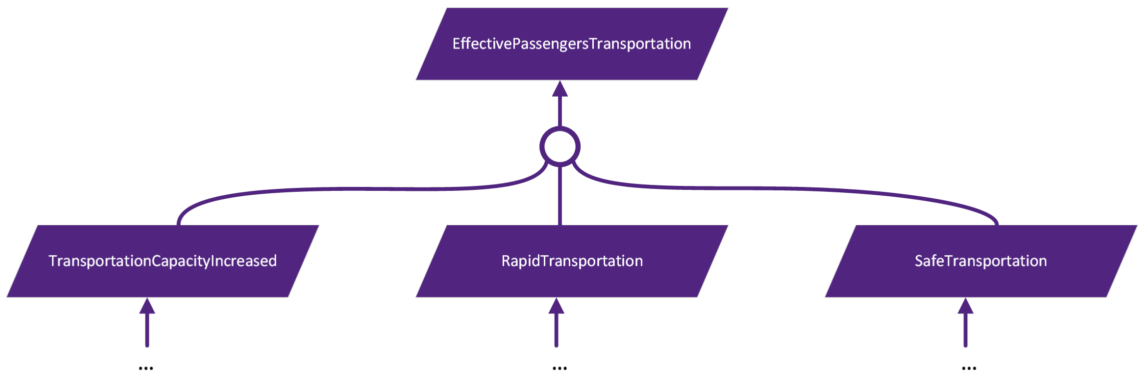

EffectivePassengersTransportation
TransportationCapacityIncreased
RapidTransportation
SafeTransportation
...
...
...

Figure 8: Example of a goal-model showing the primary goal (effective passenger transportation) and its related sub-goals (Van Lamsweerde, 2009).

2.5.2 Requirements prioritisation

Software is never finished: there is an endless number of requirements to improve software products or to increase its usefulness or to create a competitive advantage. Picking the right requirements can lead to great success while picking the wrong requirements can lead to utter failure (Berander & Andrews, 2005). The requirement prioritisation process focusses on a decision involving two requirements: which one is more important?

An informal process that is driven by intuition

The requirements prioritisation process is informal and includes invisible decisions (Lehtola et al., 2004). Continuously changing information requires that prioritisation is a continuous or iterative process. The software product manager needs to repeat this process regularly (Wiegers, 1999).

The most popular technique for requirements prioritisation is the analytical hierarchical process, followed by the quality functional deployment, planning game, and binary search tree (Achimugu et al., 2014). However, product managers hardly use these techniques, as most of them produce unreliable results or are very time-consuming. For example, the software product manager needs to manually update the prioritisation of the relevant requirements when a new requirement is introduced or deleted from the list (Achimugu et al., 2014).

> *'There is no time to analyse thousands of wishes[requirements]; much of the work is done intuitively.' (Lehtola et al., 2004)*

Criteria that influence the priority

A requirement starts with the stakeholder (typically a customer or user): an insight either motivates the stakeholder to take action (opportunity) or frustrates the stakeholder (challenge). The insight needs to contribute to achieving the product vision. Product managers should tightly couple the product vision to the value the product offers to the market (Maglyas et al., 2012). The *value* of the insight and the *vision contribution* should influence the priority of the related requirements as well.

Depending on the environment, multiple stakeholders might recognise the same insight, which increases its value for the organisation. The *reach* of the insight, therefore, should influence the priority of the related requirements.

A single requirement can rarely solve an insight on its own, and multiple goal-levels can be in between the insight and requirement (Van Lamsweerde, 2009). The team implementing the requirement needs to define the *contribution* level to the insight. At the same time, each requirement should address a small part of the insight independently. The *contribution* of a requirement to each insight should influence the priority of the requirement.

From a business perspective, the product manager needs to balance the cost of developing a requirement with the business value the requirement brings (Van Lamsweerde, 2009). The business value includes, for example, the costs of not implementing a requirement. The product manager should prefer requirements with high business value and a low cost over requirements with low business value and a high cost. The *value* and *cost* ratio should, therefore, influence the priority of a requirement.

The team implementing the requirement needs to be confident the requirement is achievable and that the requirement can address (a part of) the insight. Confidence does not only mitigate financial risks, but it also increases the efficiency of the team, and it will motivate them to implement the requirement in a way it will meet the stakeholders' expectations (Aurum & Wohlin, 2003). The *confidence* should, therefore, influence the priority of a requirement.

Relative scale

A relative scale is suitable to estimate the values of the proposed criteria (Wiegers, 1999). However, a relative scale has a disadvantage as well. The scale changes when the software product manager introduces a new requirement or removes an existing requirement. This challenge can be mitigated by trying to fit requirements into a predefined scale.

2.5.3 Alternative solution selection

Each product (or product functionality) has a goal (van Lamsweerde, 2009). A goal can range from reducing the operational costs or increasing the scalability of the product. Traditional requirements engineering modelling techniques do not include the analysis of alternative solutions (Lapouchnian, 2005) that could cause the product team to select a premature solution. The outcome of this decision is a chosen solution for reaching a specific goal. Based on this decision, the team can start the implementation.

Selecting the best possible solution

An organisation starts a software project with an agreement on *what* problem should be solved, *why* the problem needs to be solved and *who* should be involved in solving that specific problem (van Lamsweerde, 2009). For each problem, multiple (software) solutions are available. The complexity of each solution is partly defined by its functionality and by a set of non-functional requirements related to, for example, operational costs, performance, reliability, maintainability, portability, and robustness (Mylopoulos et al., 1992).

> *'Errors of omission or commission in laying down and taking properly into account such [non-functional] requirements are generally acknowledged to be among the most expensive and difficult to correct once the information system has been completed.' (Mylopoulos et al., 1992)*

Quantitative Reasoning

A solution describes the effect of the system-to-be on its surroundings. The product team defines the goal of the system-to-be and its composition, which might contain other (sub)goals or system requirements (van Lamsweerde, 2009). Out of the potential combinations of goals and requirements, the best solution needs to be selected.

Product teams can use soft goals as evaluation criteria for selecting solutions among multiple alternatives (Van Lamsweerde, 2009). A soft goal is a particular type of goal for which it is not possible to establish its reachability. It is possible to state a soft goal is more satisfied in alternative a compared to alternative b. There are two options to evaluate alternative solutions using soft goals: qualitative reasoning and quantitative reasoning (Van Lamsweerde, 2009). The disadvantage of qualitative reasoning is that the propagation rules have a high probability of generating an inconclusive outcome that does not make it suitable for this study.

> *'The aim is to determine, for each alternative, a [...] degree of satisficing of the top-level soft-goals in the goal refinement graph; the option with the best degrees of satisficing is then selected.' (Van Lamsweerde, 2009)*

Qualitative reasoning assesses the positive or negative contribution of a solution to the soft goal: the product team needs to score each potential solution against the soft goal. A score x means that the solution contributes to the soft goal for x% (Van Lamsweerde, 2009). Figure 9 presents the goal *optimal track usage*. Optimal track usage is essential for the busy rail network in, for example, Japan. The system-to-be needs to achieve this (soft) goal while taking safe transportation into account. There are two potential solutions for avoiding a train collision: avoiding trains to enter the same rail block and maintaining a worst-case stopping distance. Assuming the worst-case stopping distance is shorter than the entire rail block, this is the chosen solution. It allows maximising the rail block usage while avoiding train collisions.

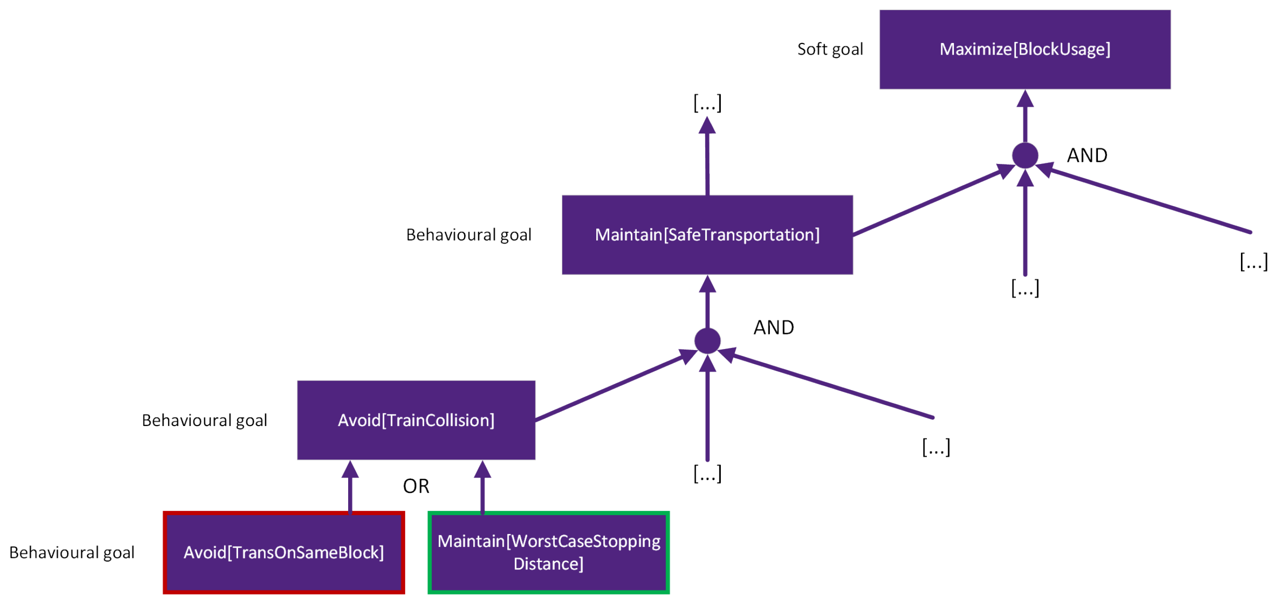


Figure 9: The selection of a solution for the efficient usage of a railway system based on higher-level soft goal (Van Lamsweerde, 2009).

We assign each soft goal with a weighted significance when the system-to-be needs to achieve multiple soft goals. Van Lamsweerde, 2009 proposes to use equation 1 to calculate the total score that sums up the individual scores of the solutions and weighs them based on relative importance.

$$totalScore(solution) = \sum_{soft\text{-}goal} (Score(solution, soft\text{-}goal) \times Weight(soft\text{-}goal)) \tag{1}$$

Equation 1 calculates the value of the combined solution and soft goal by multiplying the weighted significance of a soft goal with the score of the solution. The sum of the values assigned to a solution determines the evaluation of the solution (equation 1). Table 1 shows an example where the equation prefers to maintain the $WorstCaseStoppingDistance$ over avoiding $TrainsOnSameBlock$ (Van Lamsweerde, 2009).

Table 1: Example of the usage of equation 1 in which $Maintain[WorstCaseStoppingDistance]$ is preferred over $Avoid[TrainsOnSameBlock]$.

Soft-goal	Significance weighting	Maintain [WorstCaseStoppingDistance]	Avoid [TrainsOnSameBlock]
Maximize[BlockUsage]	0.50	0.90	0.30
Soft goal 2	0.30	0.50	0.90
Soft goal 3	0.10	0.80	0.30
Soft goal 4	0.10	0.50	1.00
Total	1.00	0.73	0.55

2.6 Conclusion

Decision-makers use more than written or scientific sources to make their decisions. The values and experience the decision-maker and organisation bring influence the decision. A decision is always time-critical. There is a balance between elaborating on the available information to increase the maturity-level and accepting the current maturity-level of the available information and making the decision. The perfect decision does not exist. There is always some prematurity in the information decision-makers have access to and limited time to increase the maturity-level.

When decision-makers know the maturity-level, they can explicitly decide to spend more time to elaborate on the information or to make the decision based on the current maturity-level.

The Semantic Web offers capabilities to validate if the information stored in an ontology is consistent, to classify information, and to detect missing information. We see an opportunity to bridge these capabilities to detect, for example, if specific information is reproducible or meeting a certain consensus-level.

Design patterns exist in different flavours but have one thing in common: they solve a general problem. Generic ontology design patterns can generalize a data structure. We have not been able to find design patterns for other Semantic Web technologies, for example, SWRL rules or SHACL constraints. Information presentation design patterns do not seem to exist literally. However, we can extract information presentation patterns from existing scientific and commercial sources.

3 Methodology

This chapter describes how we use the theoretical framework to answer the research question. We share all the source files related to this study using a GitHub repository: https://github.com/marioverhaeg/cs-thesis.

3.1 Research question

3.1.1 Scope

The results of this study should help decision-makers to move to evidence-based decision-making. This goal leads us to the main research question:

RQ: To what extent can the detection of premature information using SHACL Semantic Web constraints contribute to evidence-based decision-making?

We detect premature information using its completeness and reliability. We base the completeness of decision-relevant information on the availability of the required information and base the reliability of the decision-relevant information on the three metrics described in section 2.1.1 Decision-making model: reproducibility, consensus, and conflict. This decision meta-information allows us to calculate the information maturity-level. Section 2.1.1 Decision-making model also indicates that evidence-based decision-making depends on the quality of information. The information maturity-level measures the maturity, or quality, of information. Therefore, a higher information maturity-level contributes to evidence-based decision-making.

Ontology structure

We use Semantic Web technologies to store the information that decision-makers use to justify their decisions. First, we need to know if we can create an ontology for evidence-based management. Next, we need to ensure that we can store the information completeness and reliability in this ontology and, based on the completeness and reliability, detect premature information.

$DS1$: To what extent can we create a generic ontology design pattern for evidence-based management?

$DS2$: To what extent can we store the information completeness and reliability in an (extended) evidence-based management ontology?

$DS3$: To what extent can we use Semantic Web inferencing, consistency, and constraints to detect premature information?

Data presentation

We transform the information completeness and reliability into the information maturity-level. Therefore, we need to define the information maturity-level explicitly. Once we know when a decision-maker is interested in the information maturity-level, we can find the right way to present the information maturity-level to the decision-maker. This presentation should make it easier for the decision-maker to understand the information maturity-level and decide if the decision-relevant information is mature enough to make the decision.

$PRES1$: How do we define the maturity-level of decision-relevant information?

$PRES2$: Under which circumstances is a decision-maker interested in the maturity-level of decision-relevant information?

PRES3: To what extent can we make it easier for a decision-maker to understand the maturity-level of decision-relevant information?

3.1.2 Out of scope

The scope of this study is limited to verify if it is possible to detect premature information and to find the right way to present the information maturity-level to a decision-maker. The implementation in a production environment requires a different focus and scope. Premature information will stay premature until the decision-maker extends the ontology with new information or adjusts existing information. Extending the ontology with new information or adjusting existing information is not in the scope of this study. We also exclude an analysis of the scalability and performance of our proposals.

3.2 Scientific approach

We answer the research questions using a case-study based on the reasoning and structure proposed by de Klerk, 2018. The way we phrased the research question and the related sub-questions indicates a case study would be the most suitable method to answer these questions (Saunders et al., 2015). Figure 10 presents our conceptual approach. There is a natural iteration between the main contribution and the validation. We need to adjust a pattern if we question its suitability to solve the specified problem in the given context. When we adjust a pattern, the validation of the pattern in other contexts might require adjustments as well.

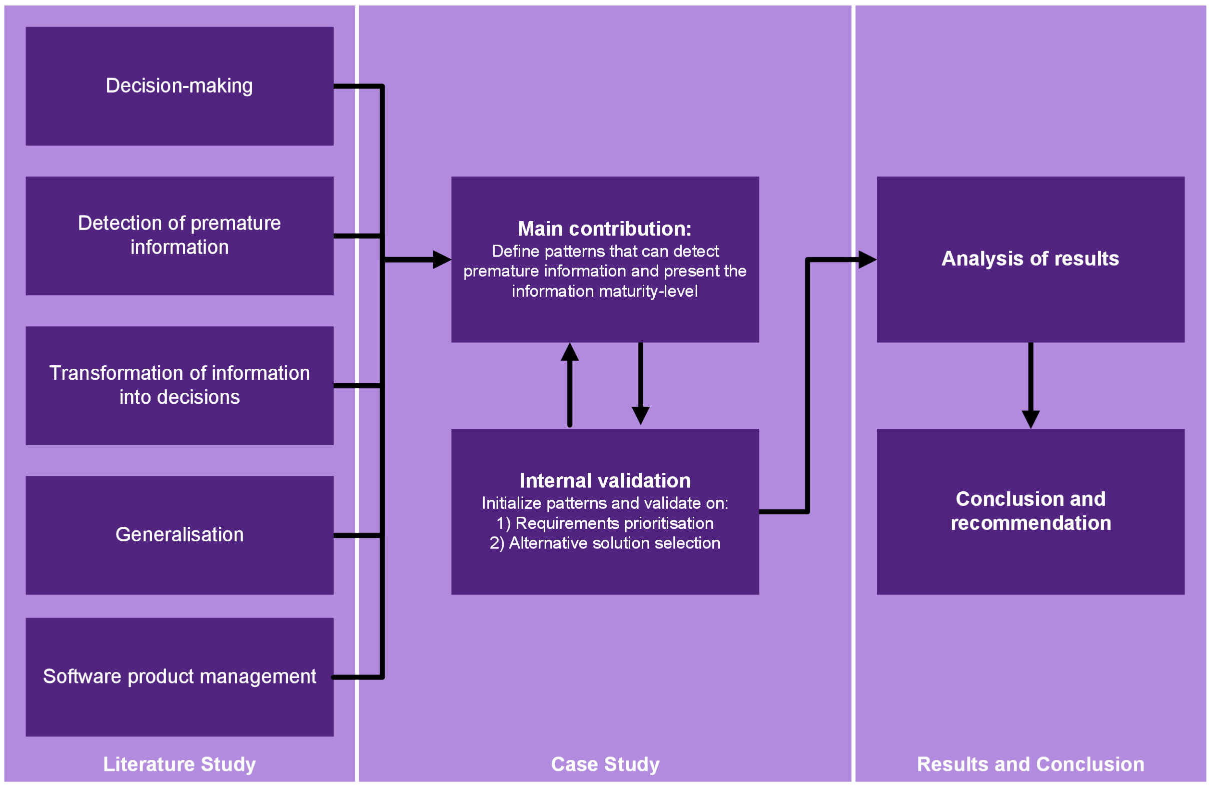

Decision-making
Detection of premature information
Transformation of information into decisions
Generalisation
Software product management
Literature Study
Main contribution:
Define patterns that can detect premature information and present the information maturity-level
Internal validation
Initialize patterns and validate on:
1) Requirements prioritisation
2) Alternative solution selection
Case Study
Analysis of results
Conclusion and recommendation
Results and Conclusion

Figure 10: The conceptual approach of this study, including the literature study, case study, and finalization phases. We re-use a structure proposed by de Klerk, 2018.

A case study confirms the proposed concept works in a specific case. A pattern solves a general problem and should, therefore, be generally applicable. We validate the pattern on two scenarios to increase our confidence that the pattern is generally applicable. The first scenario validates the pattern and includes an extensive description of its reasoning and background. We reduce the level of detail in the second scenario to prevent repetition. We indicate when we leave out details and will refer to the approach we use in the first scenario.

3.3 Activities

Table 2 presents an overview of the tools described in this section[3].

Table 2: An overview of the tools used for this study.

Name	Version	Description
Protégé	5.5.0	We use Protégé to create the base of the generic ontology design pattern, instantiating them manually and hosting the SHACL4P plugin.
SHACL4P	1.0.0	We use SHACL4P for syntax checking and validating the constraints on instantiated generic ontology design patterns.
HETS	n/a	We use a branch of HETS (*1899_gdol_parser*) that adds support for Generic DOL, which is an extension of DOL.

3.3.1 Ontology design patterns

We use generic ontology design patterns, described in section 2.4.1 Ontology design patterns, to define our ontologies formally. We use the open-source tool Protégé (the most widely used software for building and maintaining ontologies (Musen et al., 2015)) to create the base of the generic ontology design patterns and instantiate them manually. Additionally, we use HETS (the heterogeneous toolset) to validate the syntax of the generic ontology design patterns. At this moment it is not possible to use HETS to instantiate generic ontology design patterns into new ontologies or merge them into existing ontologies. We enter the ontologies into Protégé manually.

3.3.2 Detecting completeness and reliability

We define a multi-layered model using Semantic Web technologies. This model can detect the completeness, consensus, conflict, and reproducibility of information or evidence.

The first layer infers new information. Inferring information reduces the complexity of the constraints. We introduce a hierarchy of object properties and classes. When we connect two individuals using an object property in a hierarchy, the reasoner infers that the two individuals are connected by the parent of the used object property as well. Figure 11 presents an example of object property inferencing. In this case, we can define a single constraint for the abstraction layer, instead of defining multiple constraints that need to handle each case individually. Additionally, we further reduce the constraint complexity using the characteristics of an object property to infer chains of information and reduce the existing chains back into a single object property. Last, the reasoner infers class membership from the domain and range of an object property. Class membership ensures that the constraints are applied to the right information.

[3]Protégé 5.5.0 uses rdflib 3.0.0. Rdflib 3.0.0 has an issue that prevents *SPARQL COUNT DISTINCT* from working correctly (issue 404). This issue is fixed in rdflib 5.0.0, but this version is not merged into Protégé yet. We ran into this issue on several occasions and worked around it by removing the COUNT and looking at the number of results of the SPARQL query.

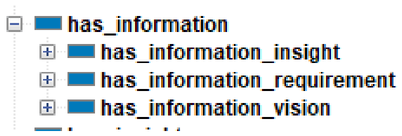


Figure 11: This example shows that the reasoner infers the object property $has\_information$ when the individual hosts the object property $has\_information\_insight$.

The second layer ensures the structural consistency of the ontology by defining that specific classes or object properties are disjoint. This definition makes the second layer especially useful to detect mistakes in the configuration of the inferencing rules.

The third layer validates the completeness and reliability of decision-relevant information:

1. We base the completeness on the existence of specific data- and object properties
2. Reproducibility equals the number of evidence sources used for decision-relevant information.
3. The consensus is equal to the number of agreements between different evidence sources.
4. The conflict is equal to the number of disagreements between different evidence sources.

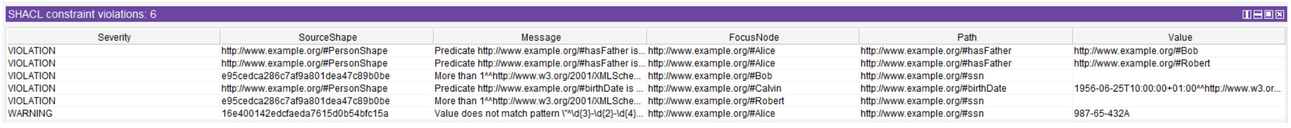
The decision design pattern defines Semantic Web constraints to detect incomplete information, the number of consensus and conflicts, and the number of evidence sources. We calculate the information maturity-level based on the number of violated constraints and the maximum number of constraints.

The Semantic Web provides two mechanisms to detect a violation of constraints: SPIN (SPARQL Rules) and SHACL. Compared to SHACL, SPIN never became a formal W3C standard, and the industry recognises SHACL as the successor of SPIN:

> *'SHACL supersedes SPIN in almost every respect. [...] Most importantly, SHACL is an official W3C Recommendation that makes it far more likely that other vendors will support it.' (Holger Knublauch, 2017)*

We select SHACL to detect the violation of constraints.

We use the Protégé plugin SHACL4P (Ekaputra & Lin, 2016) to validate the SHACL shapes. SHACL4P allows us to define constraints and validates the ontology against those constraints. Figure 12 presents an example of the output of SHACL4P.



SHACL constraint violations: 6

Severity	SourceShape	Message	FocusNode	Path	Value
VIOLATION	http://www.example.org/#PersonShape	Predicate http://www.example.org/#hasFather is...	http://www.example.org/#Alice	http://www.example.org/#hasFather	http://www.example.org/#Bob
VIOLATION	http://www.example.org/#PersonShape	Predicate http://www.example.org/#hasFather is...	http://www.example.org/#Alice	http://www.example.org/#hasFather	http://www.example.org/#Robert
VIOLATION	e95cedca286c7af9a801dea47c89b0be	More than 1^^http://www.w3.org/2001/XMLSche...	http://www.example.org/#Bob	http://www.example.org/#ssn	
VIOLATION	http://www.example.org/#PersonShape	Predicate http://www.example.org/#birthDate is ...	http://www.example.org/#Calvin	http://www.example.org/#birthDate	1956-06-25T10:00:00+01:00^^http://www.w3.or...
VIOLATION	e95cedca286c7af9a801dea47c89b0be	More than 1^^http://www.w3.org/2001/XMLSche...	http://www.example.org/#Robert	http://www.example.org/#ssn	
WARNING	16e400142edcfaeda7615d0b54bfc15a	Value does not match pattern \"^\d{3}-\d{2}-\d{4}...	http://www.example.org/#Alice	http://www.example.org/#ssn	987-65-432A

Figure 12: An example of SHACL4P output showing constraint violations in Protégé.

3.3.3 Constraint design patterns

We cannot find an explicit definition of constraint design patterns. Therefore, we define the constraints in a way they are re-usable. We add parameters to the constraints when we need those constraints in the context of the ontology design patterns. When we use parameters, we need to instantiate the constraints manually.

3.3.4 Presentation design patterns

We need to present the information maturity-level to the decision-maker. First, we extract information from the ontology to provide input to the functions that calculate the information maturity-level. Second, we select the right information presentation pattern based on the characteristics of the decision. We present the information maturity-level of the information using the selected pattern.

We build mock-ups for the data presentation pattern using generic tools, for example, Microsoft PowerPoint.

3.4 Validation

We split the validation into reproducibility (how can the results of the study be reproduced?), internal validity (how can the results of the study be validated?), and external validity (how can de results of the study be applied to other cases?).

3.4.1 Reproducibility

We ensure the reproducibility of this study using scientific literature, documented interpretation of the literature, and where applicable, documented decisions. We use, when possible, open-source tools and use our GitHub repository (https://github.com/marioverhaeg/cs-thesis) to store the generic ontology design patterns, ontologies, constraints, and queries that we present in this document. Additionally, we also store the source of this document in the GitHub repository.

3.4.2 Internal validity

The goal of the decision design pattern is to motivate decision-makers to move from intuition-based decision-making to evidence-based decision-making by detecting premature information and presenting the information maturity-level. We also want domain experts to re-use the decision design pattern when they want their decision-makers to move from intuition-based decision-making to evidence-based decision-making.

The internal validity addresses the validation of the research question based on the two scenarios described in section 2.5.2 Requirements prioritisation and 2.5.3 Alternative solution selection of the theoretical framework:

1. Scenario 1: Requirements prioritisation
2. Scenario 2: Alternative solution selection

3.4.3 External validity

The external validity addresses the validation of the research question based on the same problem in a different domain. This study focuses on a single domain (software product management). Therefore, external validation is outside of the scope of this study. We present the results of the study in a way it allows them to be applied to other domains as well.

3.4.4 Sample data

The goal of the sample data is to validate if the ontology design patterns we create are suitable to detect premature information and present the information maturity-level in requirements prioritisation and alternative solution selection. Unfortunately, it proved difficult to find sample data for requirements prioritisation and alternative solution selection. Therefore, we create sample data ourselves. We define multiple test scenarios. Each test scenario has its dedicated sample data and describes how this sample data contributes to the results of the scenario.

Creating sample data also carries a risk: we might miss validating scenarios for which we have not created sample data. We try to mitigate this risk by separating the sample data for each test case and by creating small differences in the sample data we use for different test cases.

4 Contribution

We propose the *decision design pattern* to detect premature information and present the information maturity-level to a decision-maker.

The decision design pattern detects incomplete and unreliable information and presents the information maturity-level to a decision-maker.

We define the reliability of decision-relevant information, described in section 2.1.1 Decision-making model, using the reproducibility of information, the consensus of evidence, and the conflict of evidence.

The decision-maker can decide to review the information maturity-level at any moment in time. Some decisions have deadlines, are essential, and potentially have a high impact on an organisation. Decision-makers do not make decisions with a high impact overnight, and they take their time to gather information and understand the exact impact of their decision. The information maturity-level should evolve while the decision-maker adds and changes information and the decision deadline comes closer. This way, the decision-maker uses the transparency of the information maturity-level to increase the completeness and reliability of the decision-relevant information.

4.1 Decision design pattern

We bridge the technological knowledge management strategy, described in section 2.5.2 Requirements prioritisation, to the decision design pattern. The evidence-based management pattern stores the knowledge criteria that the decision-makers require to make the decision. The decision ontology pattern uses generic ontology design patterns to define a data structure on top of the evidence-based management pattern. We add the decision presentation pattern to present the information maturity-level. Figure 13 presents the decision design pattern. We encapsulate the completeness, reproducibility, consensus, and conflict patterns into the decision ontology pattern to cover overlap in these patterns and instantiate them in a way they can function side by side.

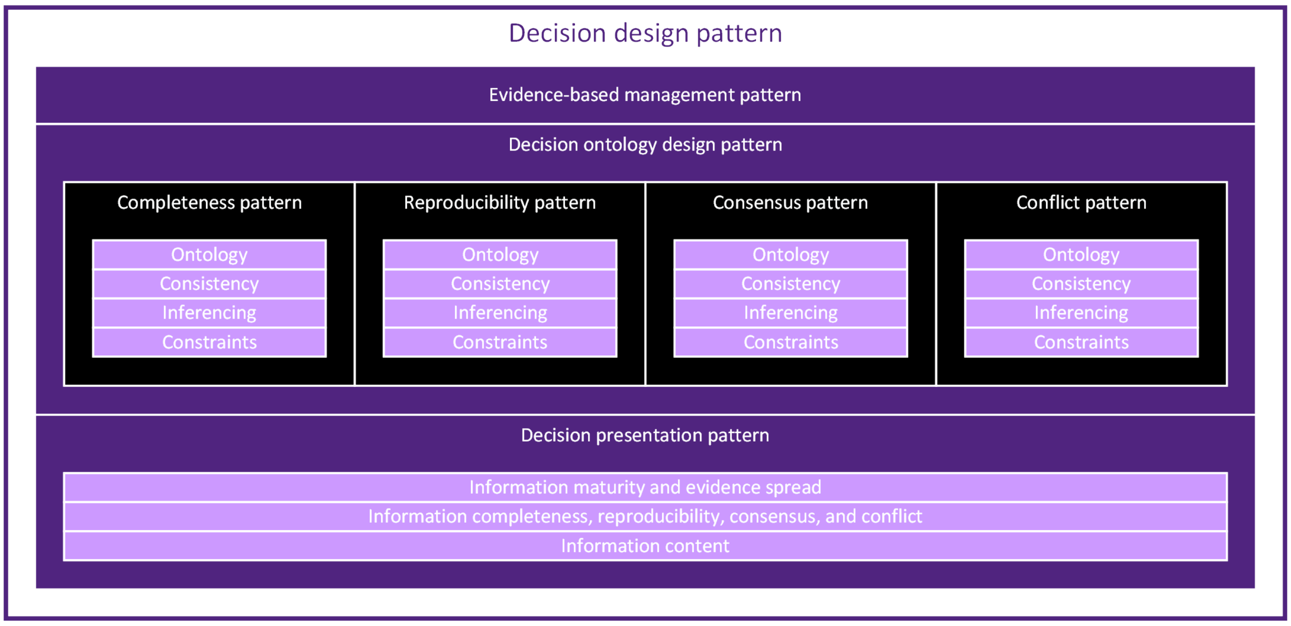

Decision design pattern
Evidence-based management pattern
Decision ontology design pattern
Completeness pattern
Reproducibility pattern
Consensus pattern
Conflict pattern
Ontology
Consistency
Inferencing
Constraints
Decision presentation pattern
Information maturity and evidence spread
Information completeness, reproducibility, consensus, and conflict
Information content

Figure 13: An overview of the decision design pattern that detects premature information and presents the information maturity-level. We encapsulate the completeness, reproducibility, consensus, and conflict patterns into the decision ontology design pattern.

We base the information maturity-level on the completeness and reliability of the information. The completeness pattern ensures that the information that a decision-maker needs to make a decision is available. The information needs to be reproducible by relating it to specific evidence sources. Additionally, we use two other patterns to measure the reliability of evidence: the conflict-level, and the consensus-level. The consolidation layer calculates the decision specific information maturity-level.

With this approach, we deviate slightly from the categories we describe in section 2.1.1 Decision-making model. We remove the information collection methodology from the scope. We capture a part of the information collection methodology in the evidence types. For example, decision-makers typically gather evidence that we classify as *Contextual_Circumstance* using observation while they can only gather *Stakeholder_Evidence* by some form of interaction with the stakeholders.

The decision-maker gets a general understanding of the information completeness and reliability using the information maturity-level and evidence-spread dashboard. We present the completeness, reproducibility, consensus, and conflict maturity-levels when the decision-maker wants to understand the information maturity-level in more details. Last, we present the completeness, reproducibility, consensus, and conflict levels per individual, including the violations. The decision-maker can use these details to increase the information maturity-level of a decision or optimise the evidence-spread.

We present multiple ontology design patterns in the next sections of this chapter. Each pattern introduces a challenge and the ontology that addresses this challenge. The included generic ontology design pattern enables domain experts to re-use the ontology in different scenarios. Additionally, we describe how we guard the consistency of the ontology, and we describe the inferred information. We present the constraints that detect premature information in the last paragraph. Last, we introduce the main challenge of the decision presentation pattern and describe how we present the information maturity-levels to the decision-maker.

Only the combination of the patterns we describe in this chapter can address the challenge captured by the research question. Therefore, we focus our validation on the combination of patterns in a specific scenario. Section 5 Validation presents the scenario-based validation that combines the patterns into a solution that addresses the challenge of the research question.

4.2 Evidence-based management pattern

4.2.1 An evidence-based management structure

Decision-makers need guidance to make decisions based on reliable information (Baba & HakemZadeh, 2012). The evidence-based management pattern provides a structure to store evidence that decision-makers can use as a source for decision-relevant information. The pattern serves as a base for other patterns that validate the information completeness and reliability.

The evidence-based management pattern supports the information completeness and reliability validation patterns by providing an evidence-based management structure.

4.2.2 Ontology

We describe four evidence types in section 2.1.1 Decision-making model: practitioner expertise and judgement, evaluated external evidence, stakeholder preferences/values, and local context (organisational actors, circumstances). We add the evidence classes *Stakeholder_Experience* and *Stakeholder_Values* as subclasses of *Stakeholder_Evidence*. We add *Stakeholder_Evidence* as a subclass to *Evidence* itself. These evidence classes slightly deviate from their original definition. We rename *practitioner* experience to *stakeholder* experience for two reasons:

1. Practitioners are, generally spoken, a subset of stakeholders: practitioners are also considered stakeholders.
2. The previous definition excluded stakeholder experience and practitioner values as evidence: if stakeholder values are considered evidence, practitioner values should also be considered evidence. This reasoning is also valid for the experience. If we consider practitioner experience as evidence, we should consider stakeholder experience as well.

We add *Contextual_Circumstances* and *Evaluated_External_Evidence* as subclasses of *Evidence*. The reasoner ensures the consistency of the ontology by defining the different evidence classes are disjoint. Figure 14 presents the structure of the pattern.

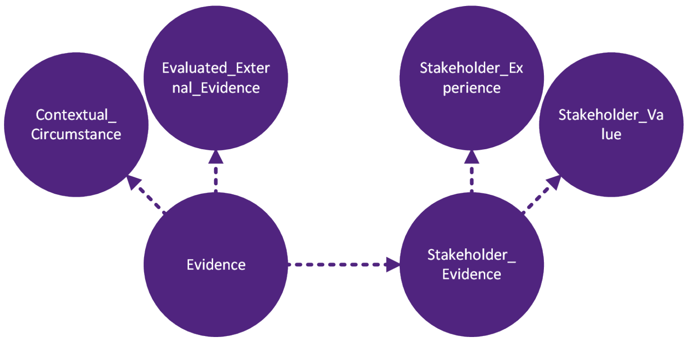
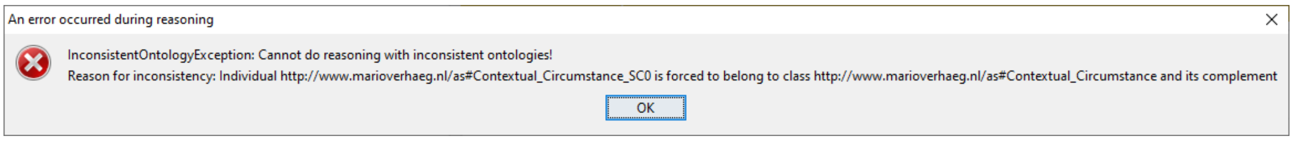


Figure 14: An overview of the evidence-based management ontology. *Stakeholder_Experience* and *Stakeholder_Values* as subclasses of *Stakeholder_Evidence*. We add *Stakeholder_Evidence* as a subclass to *Evidence* itself. Code sample 1 presents the GDOL code that instantiates this ontology.

We create these classes for two purposes. First, the evidence spread determines the reproducibility of the evidence. The decision-maker can decide to gather additional evidence to increase the reproducibility of the decision-relevant information if, for example, the decision-relevant information is only based on *Stakeholder_Experience*. Second, we extract *Contextual_Circumstances* from a written context or observation while we extract *Evaluated_External_Evidence* from other written documents. We can reproduce this evidence based on its source. For *Stakeholder_Evidence*, we need to know the stakeholder to reproduce it. Different evidence types have different behaviour and, therefore, need different classes.

Consistency

We guard the consistency of the ontology using *DisjointClasses*. *Stakeholder_Experience* is logically different from *Stakeholder_Value*. An individual cannot be *Stakeholder_Evidence*, *Contextual_Circumstance*, or *Evaluated_External_Evidence* at the same time due to the difference in definition. When the reasoner classifies an individual as two disjoint classes, it will throw an inconsistency error and cannot continue reasoning. Figure 15 presents an example of an inconsistency error in Protégé. Code sample 1 presents the implementation of the *DisjointClasses*.



```
An error occurred during reasoning

InconsistentOntologyException: Cannot do reasoning with inconsistent ontologies!
Reason for inconsistency: Individual http://www.marioverhaeg.nl/as#Contextual_Circumstance_SC0 is forced to belong to class http://www.marioverhaeg.nl/as#Contextual_Circumstance and its complement
```

Figure 15: An inconsistent ontology structure that a reasoner detects in Protégé. In this case, the individual *Contextual_Circumstance_SC0* belongs to the classes *Contextual_Circumstance* and *Evaluated_External_Evidence*. *Contextual_Circumstance_SC0* and *Evaluated_External_Evidence* are disjoint classes. The reasoner does not accept that *Contextual_Circumstance_SC0* belongs to both of these classes.

Inferencing

The evidence-based management pattern uses a simple structure of classes and does not use inferencing to increase the amount of information in the ontology.

Generic Ontology Design Pattern

Code sample 1 presents the generic ontology design pattern of the evidence-based management pattern using GDOL. We instantiate the evidence-based management pattern once per ontology it needs to extend. The evidence-based management pattern does not require parameters; therefore, we instantiate an *ontology* instead of a *pattern*.

```
ontology EBM =
 Class: Evidence
 Class: Stakeholder
 Class: Stakeholder_Evidence SubClassOf: Evidence
 Class: Stakeholder_Experience SubClassOf: Stakeholder_Evidence
 Class: Stakeholder_Value SubClassOf: Stakeholder_Evidence
 Class: Contextual_Circumstance SubClassOf: Evidence
 Class: Evaluated_External_Evidence SubClassOf: Evidence
 DisjointClasses: Stakeholder_Evidence, Contextual_Circumstance,
    Evaluated_External_Evidence
 DisjointClasses: Stakeholder_Experience, Stakeholder_Value
```

Code sample 1: The GDOL code that instantiates the generic ontology design pattern for evidence-based management. The instantiation includes the instantiation of the classes as well as the characteristics of those classes. Figure 14 presents the structure of the pattern.

Constraints

It is possible to define constraints based on the stake of a specific evidence type. For example, the *Evaluated_External_Evidence* should represent 20% of the evidence related to a specific decision. However, these constraints would be very context-sensitive. We leave it up to the decision-maker to evaluate the mix of evidence-types manually on a case-by-case basis. Alternatively, the domain expert can manually add constraints based on the preference of the organisation implementing the decision design pattern.

4.3 Decision ontology pattern

4.3.1 An information maturity-level structure

The evidence-based management, completeness, reproducibility, consensus, and conflict patterns solve separate generic problems. At the same time, these patterns overlap in their ontology structure. This overlap makes it difficult to use these patterns in the same environment. The decision design pattern provides the glue between the completeness, reproducibility, consensus, and conflict patterns, using the evidence-based management pattern as a base.

The decision presentation pattern combines the output from the completeness, reproducibility, consensus, and conflict patterns to calculate the *information maturity-level* for a decision. The decision-maker needs to make a *meta-decision* based on the information maturity-level:

1. If the information maturity-level is acceptable, the decision-maker can make the main-decision.
2. If the information maturity-level is not acceptable, the decision-maker needs to increase the information maturity-level until it is acceptable.

The outcome of the meta-decision depends on the complexity and impact of the main-decision. If the complexity and impact of the main-decision are low, a lower information maturity-level might be acceptable. However, if the impact and complexity of the main-decision are high, we expect that the requirements towards the information maturity-level are higher as well.

The decision ontology pattern increases the transparency of the completeness and reliability of decision-relevant information.

The scale of the impact and complexity of a decision are organisation dependent. For example, a C-level decision in an organisation with 3000 employees has a different impact compared to a C-level decision in an organisation with 300 employees. As a result, we cannot automate this meta-decision. In essence, a human needs to evaluate, based on knowledge and experience, if the complexity and impact of the decision justify the information maturity-level.

4.3.2 Completeness

Detecting completeness

A decision-maker makes a decision knowing context-relevant information for that specific decision. Some information crucial for decision x might be irrelevant for decision y. Each decision requires different context-relevant information. This pattern allows a domain expert to define the context-relevant information.

The completeness pattern validates the completeness of information by detecting missing information.

Ontology

The completeness pattern detects if an individual classified as C is incomplete, considering a required property p. We achieve this by creating a data property or an object property and use parameters to define its domain and range. The constraints detect individuals classified as C that do not host the data or object property. The detected individuals are considered incomplete and, therefore, premature.

Inferencing

The completeness pattern infers the class of individuals from the domain or range of the data or object property. For example, if $Information$ is the domain of the property $data\_description$, then the reasoner will infer individuals that have a $data\_description$ as $Information$. Figure 16 presents this example in Protégé.

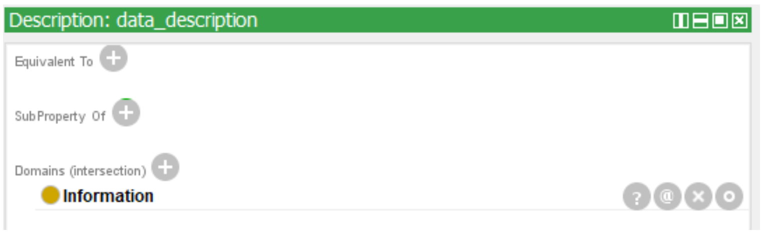


Figure 16: If $Information$ is the domain of the property $data\_description$, then the reasoner will infer individuals that have a $data\_description$ as $Information$.

Inconsistency

Code sample 2 presents the generic ontology design pattern to create a data property. This code sample includes a range. The definition of the range limits the data range that the data property accepts, for example, a $xsd : string$ accepts strings or $xsd : int$ accepts integers. The ontology is inconsistent when the data property stores a value that is outside of the defined range.

Generic ontology design pattern

Code sample 2 adds a data property or an object property to an existing class. We use three parameters to instantiate the data or object property: c defines the class that should host the data property, p defines the data property itself, and r defines the range of the data property. We use the range of the data property to restrict its content using, for example, regular expressions and use the range of the object property to infer the class of an individual. Figure 17 presents the data property, and figure 18 presents the object property.

```
pattern Completeness_dp [Class: c; DataProperty: i; Datatype: r] =
   DataProperty: i Domain: c Range: r
pattern Completeness_op [Class: c; ObjectProperty: i; Datatype: r] =
   ObjectProperty: i Domain: c Range: r
```

Code sample 2: The GDOL code for adding a required data property to an existing class using two parameters. We use c, i, and r as parameters to instantiate the data or object property.

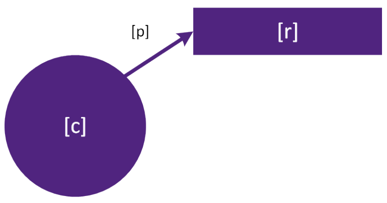


Figure 17: The completeness of a class using a data property. Code sample 2 presents the GDOL code for adding a data property to an existing class. Code sample 2 presents the GDOL code that instantiates this ontology.

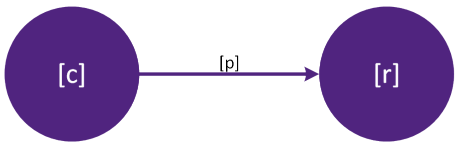


Figure 18: The completeness of a class using an object property. Code sample 2 presents the GDOL code for adding an object property to an existing class. Code sample 2 presents the GDOL code that instantiates this ontology.

Constraints

The SHACL shape in code sample 3 detects when an individual classified as c does not host the defined data or object property p. Each individual classified as c should have at least one path (object or data property) p. The SHACL shape monitors the existence of the data or object property using the cardinality constraint $minCount$. c and p set the context of the constraints.

```
[c]Shape a sh:NodeShape;
   sh:targetClass [c];
   sh:property [
      sh:path [p];
      sh:severity sh:Violation;
      sh:minCount 1;
      sh:message "Completeness: add [c] to [p]."; ]
```

Code sample 3: The SHACL code that validates if the individuals classified as c host the property p. We use the cardinality constraint $sh : minCount$ for this detection: each individual classified as c should have at least one path (object or data property) p.

4.3.3 Reproducibility

Detecting reproducibility

We use reproducibility to evaluate the reliability of decision-relevant information. The reproducibility pattern can detect when information cannot be traced back to an evidence source. When evidence cannot be traced back to an evidence source, the pattern detects the information as premature.

The reproducibility pattern validates the information reliability by detecting when information cannot be traced back to an evidence source.

Ontology

Information is reproducible when it can be traced back to evidence or an information source, for example, a contextual circumstance or the value of a stakeholder. This pattern relates information to an evidence source using an object property. An information class can be based on another information class as well, as long as the chain of information is evidence-based. We use the completeness pattern to ensure the required data properties are available. Figure 19 presents a chain that connects information to evidence. The *based_on_information* object property is transitive. When we base information a on information b, and information b on information c, then information a is also based on information c. We need the transitive characteristic to query the evidence sources of information for the decision presentation pattern.

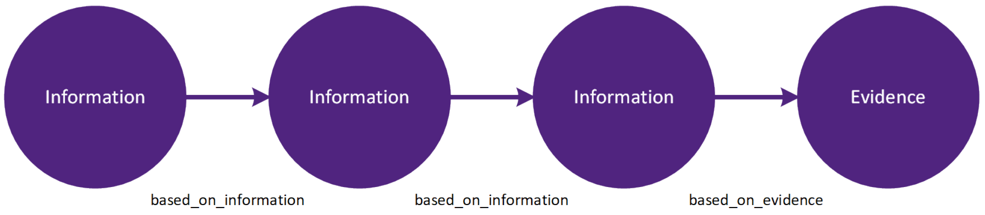


Figure 19: An example of an evidence-based chain of information. The reproducibility pattern should detect if the information in the chain is not evidence-based.

We extend the evidence-based management pattern in two ways:

1. Contextual circumstances and evaluated external evidence naturally refer to their actual evidence source, for example, a scientific article or an observation. Stakeholder evidence requires a specific stakeholder as a source of evidence. We extend the evidence-based management pattern with one class (*Stakeholder*) and the related object property (*shared_by*).
2. Individuals (classified as *Information*) hosting data properties should be evidence-based or information-based. The object property *based_on_evidence* relates a class to the evidence class. The object property *based_on_information* relates a class to an information class.

Figure 20 presents the resulting ontology. We have marked the extensions of the evidence-based management pattern in blue.

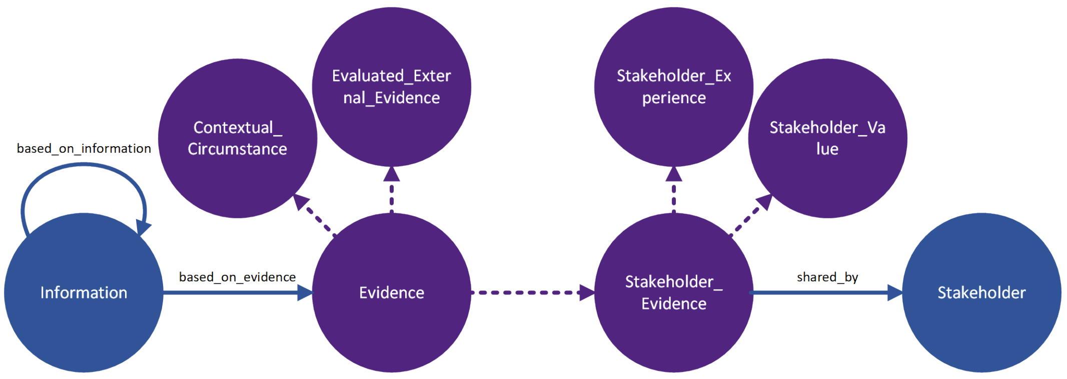


Figure 20: The reproducibility ontology. We have marked the extensions of the evidence-based management pattern in blue. Code sample 4 presents the GDOL code that we use to instantiate this pattern.

Inferencing

We use the domain and range of the *based_on_information*, *based_on_evidence*, and *shared_by* object properties to allow the reasoner to classify individuals that are using these object properties. Table 3 presents the classification the reasoner infers from the domain and range configuration.

Table 3: We use the domain and range of the *based_on_information*, *based_on_evidence*, and *shared_by* object properties to allow the reasoner to classify individuals that are using these object properties.

Domain	Object property	Range
Information	*based_on_information*	*Information*
Information	*based_on_evidence*	*Evidence*
Stakeholder_Evidence	*shared_by*	*Stakeholder*

Inconsistency

We guard the consistency of the ontology using *DisjointClasses*. The reasoner cannot classify an individual as *Information* and *Evidence* at the same time. If we allow the reasoner to do this, it might create a circular dependency in the reproducibility chain figure 19 presents. *Information* typically represents a statement that we can trace back to at least one *Evidence* source. Code sample 4 presents the implementation of the *DisjointClasses*.

Generic ontology design pattern

Figure 20 presents an ontology that extends the evidence-based management ontology and enables the detection of unreproducible information. Code sample 4 presents the GDOL code that extends the evidence-based management ontology.

```
ontology Reproducibility_Basic = EBM then
 Class: Stakeholder
 Class: Information
 ObjectProperty shared_by Domain: Stakeholder_Evidence Range: Stakeholder
 ObjectProperty based_on_evidence Domain: Information Range: Evidence
 ObjectProperty based_on_information Domain: Information Range: Information
    Characteristics: Transitive
 DisjointClasses: Information, Evidence
```

Code sample 4: The GDOL code for relating information to evidence and stakeholder evidence to a stakeholder. We introduce two new classes (*Stakeholder* and *Information*) and three new object properties (*shared_by*, *based_on_evidence*, and *based_on_information*. Additionally, we define that *Information* and *Evidence* are disjoint. Figure 20 visualises the result of executing the code.

The classes that store decision-relevant information are required to be a subclass of *Information*. Code sample 5 presents the GDOL code that relates information to evidence. The inferencing using the domain and range of the *based_on_information*, *based_on_evidence*, and *shared_by* object properties contributes to this classification as well.

```
pattern Reproducibility_Context [Class: dri; Class: i] =
 Class: [dri] SubClassOf: [i]
```

Code sample 5: The GDOL code that classifies individuals as a sub-class of *Information*. We use *dri* (decision relevant information) and *i* (information) as parameters.

Constraints

Code sample 6 presents the SHACL shape that detects unreproducible information. Each individual that is classified as *Information* is required to host the object property *based_on_information* or *based_on_evidence*. Individuals that are classified as *Information* can be traced back to an evidence or information source by hosting one of these two object properties. If the individual does not host one of the object properties, its information cannot be traced back to an evidence source, and the pattern detects the information as premature. The SHACL shape monitors the existence of the object properties using the cardinality constraint *minCount*.

```
Used_InformationShape a sh:NodeShape;
   sh:targetClass Information;
   sh:property [
      sh:or (
         [sh:path based_on_information; sh:minCount 1;]
         [sh:path based_on_evidence; sh:minCount 1;]
      )
      sh:severity sh:Violation;
      sh:message "Reproducibility: enter an information or evidence source for this
    information."; ];
```

Code sample 6: The SHACL code that detects if *Information* is not *based_on_evidence* or *based_on_information*. The SHACL shape monitors the existence of these object properties using the cardinality constraint *minCount*.

Code sample 7 presents the SHACL shape that detects when the pattern cannot trace back *Stakeholder_Experience* or a *Stakeholder_Value* to a *Stakeholder*. We use the *shared_by* combined with the cardinality constraints *sh* : *minCount* object property to achieve this. The constraints generate a violation if an individual that is classified as *Stakeholder_Evidence* is not *shared_by* a stakeholder.

```
StakeholderShape a sh:NodeShape;
   sh:targetClass Stakeholder_Evidence;
   sh:property [
      sh:path shared_by;
      sh:severity sh:Violation;
      sh:minCount 1;
      sh:message "Reproducibility: enter a stakeholder that serves as the source of this
    stakeholder evidence."; ];
```

Code sample 7: The SHACL code that detects when a stakeholder does not share *Stakeholder_Evidence*. We use the cardinality constraint *sh* : *minCount* for this detection: each individual classified as *Stakeholder_Evidence* should have at least one path *shared_by*. The range of *shared_by* is *Stakeholder*.

4.3.4 Consensus

Detecting consensus

Information that is based on multiple evidence sources is more reliable than information that is based on one evidence source. For example, one scientific paper supported by stakeholder experience is more reliable than a scientific paper alone.

The consensus pattern validates the evidence reliability by detecting when evidence does not agree with at least one additional evidence source.

Ontology

We define consent evidence as evidence that agrees with at least one other evidence source. The reproducibility pattern provides the structure on which we create the consensus pattern. We introduce the *agrees_with* object property. Figure 21 presents the consensus pattern. We mark the extension of the reproducibility pattern in green. For example, if two stakeholders have the same experience, the *agrees_with* object property can link this stakeholder experience.

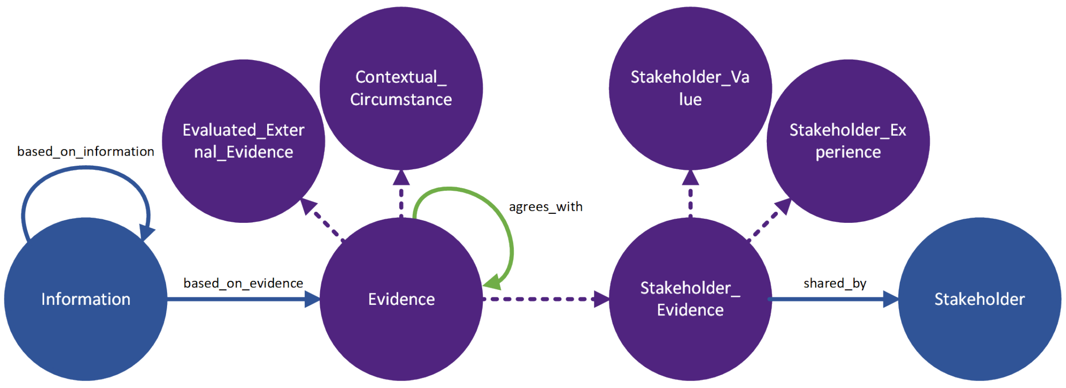
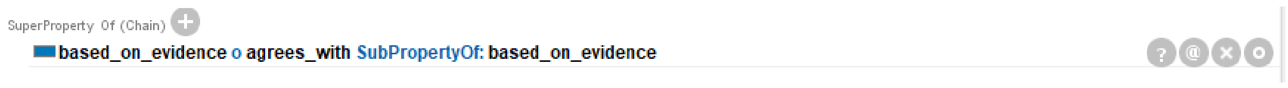


Figure 21: The consensus pattern that we have based on the reproducibility pattern. We have extended the reproducibility pattern with the object property *agrees_with*. Code sample 8 extends the reproducibility pattern using GDOL.

Inferencing

We can measure the consensus-level of an evidence source using the *agrees_with* object property. We extend the *based_on_evidence* object property with a super property. When we base *InformationX* on *EvidenceA*, and *EvidenceA* agrees with *EvidenceB*, the reasoner bases *InformationX* on *EvidenceB* as well. Figure 22 presents the super property that infers this knowledge from the existing ontology structure.


SuperProperty Of (Chain)

based_on_evidence o agrees_with SubPropertyOf: based_on_evidence

Figure 22: The super property that infers the object property *based_on_evidence* in Protégé.

The *agrees_with* object property is symmetric. If individual *A* agrees with individual *B*, individual *B* should also agree with individual *A*. Figure 23 presents the characteristics of the *agrees_with* object property.

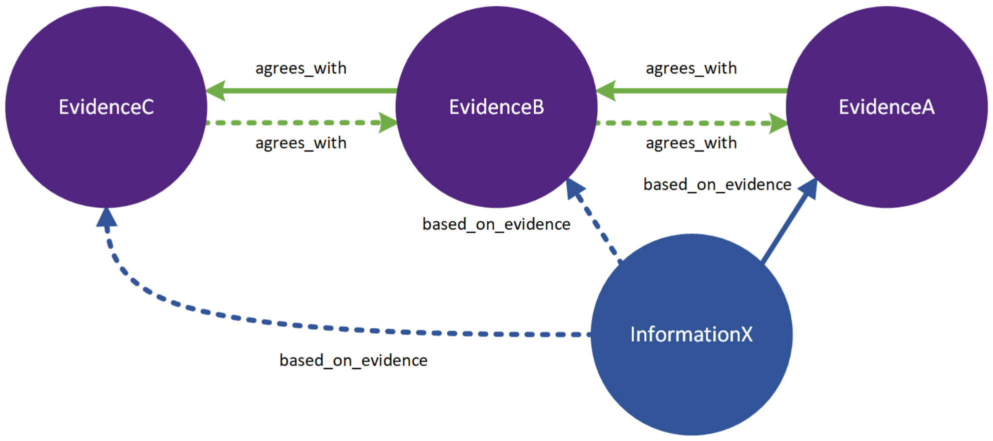


Figure 23: An example of the impact of the symmetric characteristics of the *agrees_with* object property. The reasoner infers the dotted lines based on the super property figure 22 presents and the symmetric characteristics of *agrees_with*.

Inconsistency

The consensus pattern inherits the consistency validation from the reproducibility pattern. We describe the consistency of the reproducibility pattern in section 4.3.3 Inconsistency.

Generic ontology design pattern

Code sample 8 presents the described solution into GDOL. We take the *Reproducibility_Basic* pattern, introduce the *agrees_with* object property, and extend the *based_on_evidence* object property.

```
ontology Consensus = Reproducibility_Basic then
 ObjectProperty: agrees_with Domain: Evidence Range: Evidence Characteristics: Symmetric
 ObjectProperty: based_on_evidence SubPropertyChain: based_on_evidence o agrees_with
    SubPropertyOf: based_on_evidence
```

Code sample 8: The GDOL code that extends the reproducibility pattern and results in the consensus pattern. We take the *Reproducibility_Basic* pattern, introduce the *agrees_with* object property, and extend the *based_on_evidence* object property. Figure 21 presents the generic ontology design pattern.

Constraints

We define one consensus level for each individual classified as *Evidence*. We set the consensus-level to 1. This consensus level ensures that each used evidence source has at least one other agreeable evidence source. The minimum consensus level can be adjusted depending on the environment. For example, in life-safety environments, the consensus level might be increased. We use the cardinality constraint *sh* : *minCount* for the consensus detection: each individual classified as *Evidence* should have at least one path *agrees_with*.

```
UsedEvidenceShape a sh:NodeShape;
   sh:targetClass Evidence;
   sh:property [
      sh:path agrees_with;
      sh:severity sh:Violation;
      sh:minCount 1;
      sh:message "Consensus: ensure the evidence is in agreement with at least one
    additional evidence source."; ];
```

Code sample 9: The SHACL shapes that detect when information does not meet the minimum consensus level.

4.3.5 Conflict

Detecting conflict

Information that has conflicting evidence sources is less reliable compared to information that does not have conflicting evidence sources. For example, stakeholder experience that is conflicting with evaluated external evidence is less reliable compared to stakeholder experience that does not have any conflict. We define the number of evidence sources that causes a conflict with the decision-relevant information as the level of conflict. We define the level of conflict using the reproducibility of information. When information is reproducible by one evidence source, and this evidence source has one additional conflicting evidence source, the level of conflict is 1. When the evidence source has two conflicting evidence sources, the level of conflict is 2. Compared to a fuzzy approach that can classify certain conflict levels as, for example, *low*, *medium*, or *high*, the number-based approach makes it easier to compare conflict levels. We do not compare conflict-levels in this study.

The conflict pattern validates information reliability by detecting conflicting evidence.

Ontology

We define conflicting evidence as evidence that disagrees with another evidence source. The reproducibility pattern serves as a natural base for the conflict pattern. The reproducibility pattern provides the structure on which we create the conflict pattern using the object property *disagrees_with*. Figure 24 presents the conflict pattern. We mark the extension of the conflict pattern in red. We measure the level of conflict between evidence sources using the *disagrees_with* object property. For example, if two stakeholders have a different experience, the *disagrees_with* object property represents this conflicting stakeholder experience.

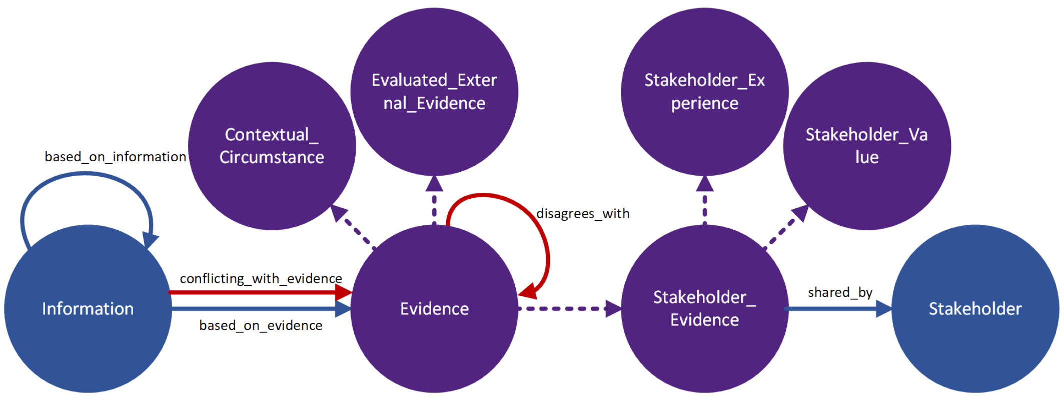
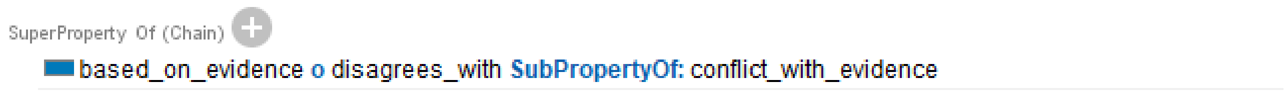


Figure 24: The conflict pattern uses the reproducibility pattern and extends it with the object property *disagrees_with*. Code sample 10 presents the GDOL code for instantiating the conflict pattern.

Inferencing

We introduce an additional object property: *conflict_with_evidence*. When we base *InformationX* on *EvidenceA*, and *EvidenceA* disagrees with *EvidenceB*, *InformationX* conflicts with *EvidenceB*. This conflict reduces trust in *InformationX*. Figure 25 presents the super property that infers this knowledge from the existing ontology structure.


SuperProperty Of (Chain)

based_on_evidence o disagrees_with SubPropertyOf: conflict_with_evidence

Figure 25: The configuration of the super property that infers the object property *conflict_with_evidence* in Protégé.

disagrees_with is symmetric and irreflexive. Disagreement is always symmetric: if *StakeholderA* disagrees with *StakeholderB* on a specific subject, then *StakeholderB* also disagrees with *StakeholderA*. Figure 26 presents the situation in which *disagrees_with* is symmetric and irreflexive.

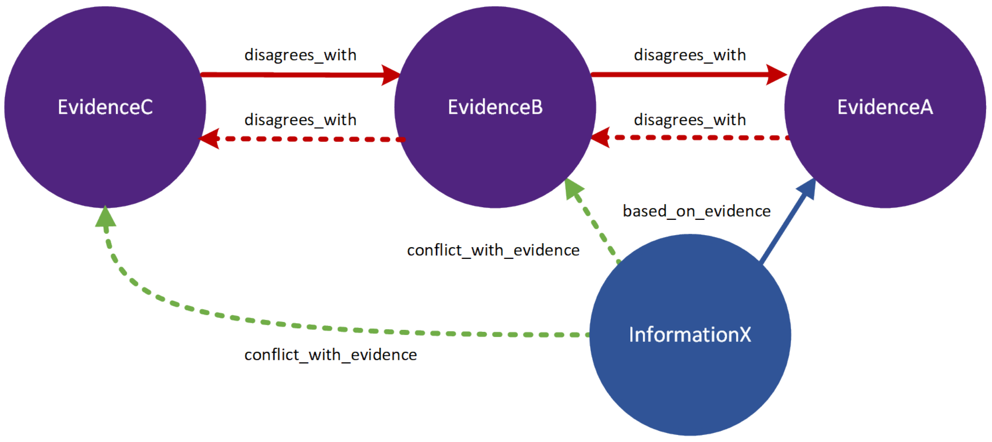


Figure 26: The symmetric and irreflexive characteristics of *disagrees_with*. The reasoner infers the dotted lines based on the super property figure 25 presents.

Inconsistency

The consensus pattern inherits the consistency validation from the reproducibility pattern. We describe the consistency of the reproducibility pattern in section 4.3.3 Inconsistency.

Figure 27 presents the hypothetical situation in which *disagrees_with* is symmetric, transitive, and irreflexive. This situation results in an inconsistent ontology. The reasoner infers the *disagrees_with* object property on $EvidenceA$ based on the transitivity characteristic. This object property causes conflict with the irreflexive character of $disagrees\_with$. The transitivity helps us to discover disagreement chains and ensures we present the right conflict level to the decision-maker. Transitivity causes a problem: evidence cannot disagree with itself. When we combine the irreflexive and transitive characteristics, the reasoner might infer the $disagrees\_with$ object property on an evidence source. For example, we assume R is transitive and irreflexive. By transitivity, we conclude $xRy \wedge yRx \rightarrow xRx$. However, this is not possible as R should be irreflexive as well.

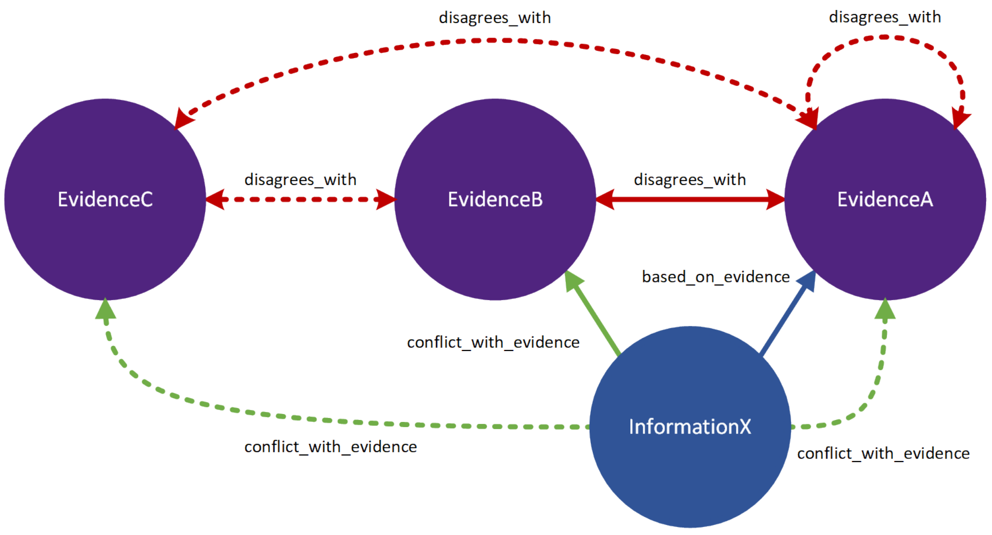


Figure 27: The transitive characteristic of $disagrees\_with$ causes an inconsistent ontology. The reasoner infers the dotted lines based on the super property figure 25 presents.

Generic ontology design pattern

Code sample 10 presents the described solution into GDOL. We take the $Reproducibility\_Basic$ pattern and add the $disagrees\_with$ and $conflict\_with\_evidence$ object properties.

```
ontology Conflict = Reproducibility_Basic then
  ObjectProperty: disagrees_with Domain: Evidence Range: Evidence Characteristics:
    Symmetric, Irreflexive
  ObjectProperty: conflict_with_evidence SubPropertyChain: based_on_evidence o
    disagrees_with SubPropertyOf: conflict_with_evidence
end
```

Code sample 10: The GDOL code that extends the reproducibility pattern and results in the conflict pattern. We take the $Reproducibility\_Basic$ pattern and add the $disagrees\_with$ and $conflict\_with\_evidence$ object properties.

Constraints

The conflict pattern requires the definition of a conflict level. Conflict reduces the reliability of the information and evidence. Therefore, we do not accept any conflict for our evidence and define the maximum conflict-level as 0. Any conflict that the constraints detect immediately causes a violation. We use the cardinality constraint $sh : minCount$ for this detection: each individual classified as $Information$ should not have any path (object property) $conflict\_with\_evidence$.

4.3.6 Pattern consolidation

The decision ontology pattern reduces the complexity of the instantiation of the completeness, reproducibility, consensus, and conflict patterns.

```
Used_InformationShape a sh:NodeShape;
   sh:targetClass Used_Information;
   sh:property [
      sh:path conflict_with_evidence;
      sh:severity sh:Violation;
      sh:maxCount 0;
      sh:message "Conflict detected. Please resolve the conflict or re-consider using
    this information."; ];
```

Code sample 11: The SHACL shapes that detect if there is conflict.

Pattern dependencies

We cannot reproduce information that does not exist. We validate the reproducibility of existing, and therefore, complete information. The consensus and conflict patterns have a similar dependency on the reproducibility pattern. When information is evidence-based, we can validate if the evidence agrees with other evidence (and define the level of consensus) or if information disagrees with other evidence (and define the level of conflict).

There is no consensus or conflict without reproducibility. There is no reproducibility without completeness.

Completeness and reproducibility using N-ary relations

The completeness pattern validates the completeness of data properties and object properties. The reproducibility pattern validates the reproducibility of individuals using the object properties *based_on_evidence* and *based_on_information*. We need to relate a data property, containing decision-relevant information, to an object property. Figure 28 presents an example: *dataproperty*1 would be reproducible based on *based_on_evidence*. However, *dataproperty*2 would not be reproducible.

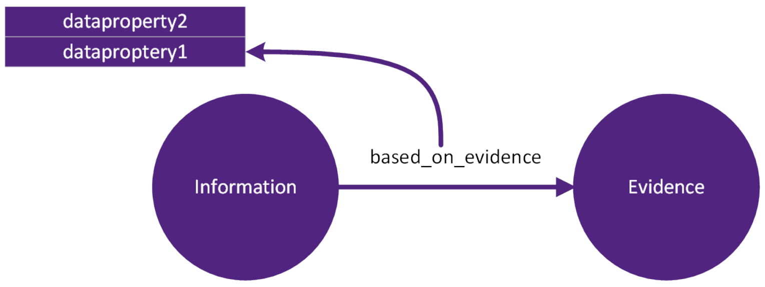
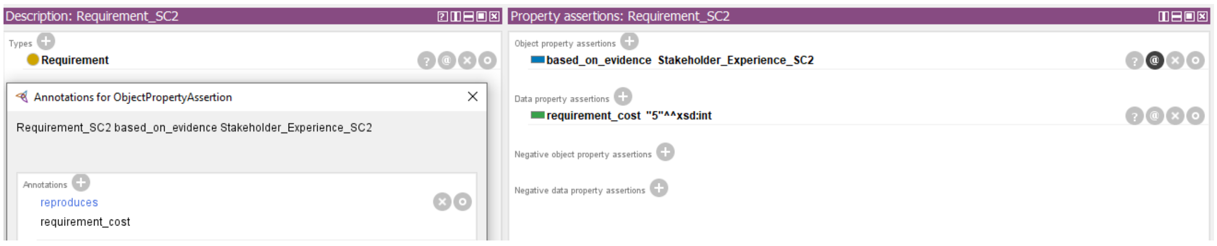


Figure 28: How can we reproduce *dataproperty*1 based on the object property *based_on_evidence*? However, *dataproperty*2 is not reproducible.

A property is a binary relation that relates two individuals (Noy & Rector, 2006). This concept makes it challenging to describe a relationship. In our case, we would describe the *based_on_evidence* relationship with the data properties this relationship represents. We can describe object properties using annotations. Figure 29 presents the implementation of the annotation on the object property *based_on_evidence* in Protégé, indicating that it *reproduces* the *requirement_cost*.


Description: Requirement_SC2

Types
Requirement

Annotations for ObjectPropertyAssertion

Requirement_SC2 based_on_evidence Stakeholder_Experience_SC2

Annotations
reproduces
requirement_cost

Property assertions: Requirement_SC2

Object property assertions
based_on_evidence Stakeholder_Experience_SC2

Data property assertions
requirement_cost "5"^^xsd:int

Negative object property assertions

Negative data property assertions

Figure 29: The implementation of the annotation on the object property *based_on_evidence* in Protégé, indicating that it *reproduces* the *requirement_cost*.

However, annotation properties are merely descriptive and cannot define a characteristic of an object or data property (Bechhofer et al., 2004). Therefore, the annotation properties are not suitable for inferencing or constraint validation. Additionally, we cannot use annotation properties to validate the reproducibility.

We define n-ary relations to solve this challenge (Noy & Rector, 2006). An n-ary relation introduces a new class for information that we would typically store in a data property. The new class hosts the actual information as data properties. We define three types of individuals for our purpose: *root*, *information*, and *target*. The *root* provides the context for the *information*. The *information* stores the actual information in two data properties (*data_value* and *data_description*), and the *target* serves as the target for reproduction. We use the *data_value* to give an integer to the information. The *data_description* describes the value. We suggest setting the *data_value* to 0 when the context does not require the use of the *data_value*. Figure 30 presents the definitions.

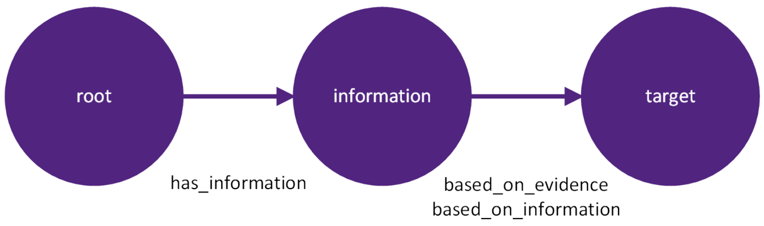


Figure 30: The *root*, *information*, and *target* individuals in n-ary relation.

Figure 31 presents an example of this concept. Class r, the root class, has two decision-relevant information properties: $n1$ and $n2$. These decision-relevant information properties have at least the *data_value* and *data_description* data properties. We classify them as *Information*. $n1$ and $n2$ are reproducible using the object properties *based_on_evidence* or *based_on_information*. We need to ensure that the completeness and reproducibility patterns function in this environment.

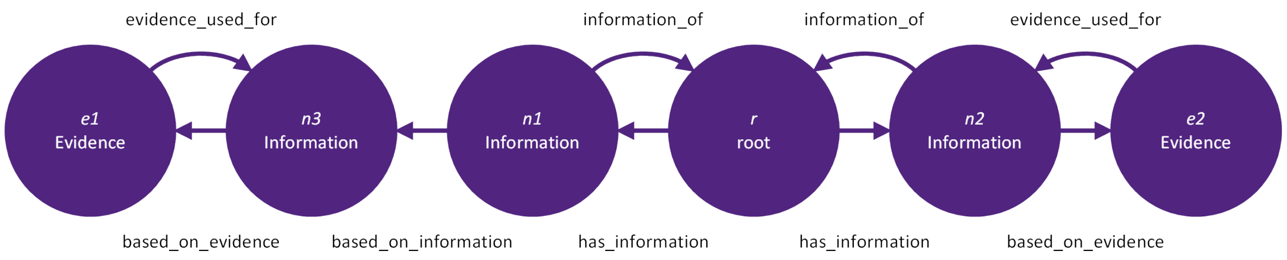
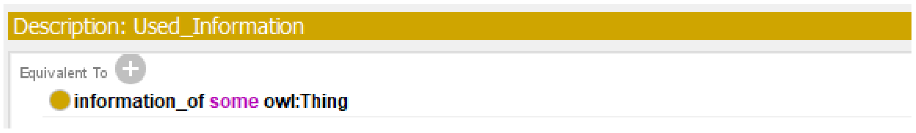


Figure 31: Class r with two decision-relevant information properties: $n1$ and $n2$. These decision-relevant information properties have a *data_value* and a *data_description*. $n1$ and $n2$ are reproducible using the object property *based_on_evidence*.

We validate the completeness of information using the $has\_information$ object property, including its range. We validate the reproducibility of information using the *based_on_evidence* and *based_on_information* object properties.

Used classes

We use $Used\_*$ classes throughout the patterns to define the scope of the constraints. Each $Used\_*$ class is a subclass of its parent. We define the $Used\_*$ class in a way the reasoner classifies individuals that are relevant for the constraints. Figure 32 presents, for example, the $Used\_Information$ class. There might be a lot of individuals classified as $Information$. However, these individuals are only relevant to the decision-maker if the decision-maker uses them. The reasoner classifies individuals as $Used\_Information$ when they have the $information\_of$ object property. We use the inverse of the $has\_information\_*$ object property to infer that information is used in the context of a decision-relevant root individual. We use the $evidence\_used\_for$ object property to infer $Used\_Evidence$ similarly.


Description: Used_Information

Equivalent To

information_of some owl:Thing

Figure 32: The $Used\_Information$ configuration in Protégé. The reasoner classifies individuals as $Used\_Information$ when they have the $information\_of$ object property.

Inferencing

We use inferencing in the decision design pattern to infer the information types from the object property $has\_information\_*$. The completeness pattern uses the information generated by the reasoner to validate the existence of specific data properties. Figure 33 presents an example in which individual A is connected to individual B by $has\_information\_temperature$. We set the range of $has\_information\_temperature$ to the class $Temperature$. As a result, the reasoner infers that B must be of type $Temperature$. This mechanism allows the completeness pattern to validate if individuals classified as $Temperature$ have specific data or object properties.

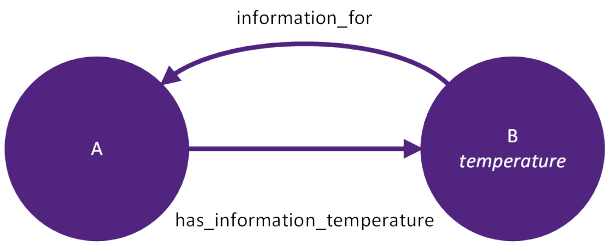


Figure 33: An example in which individual A is connected to individual B by $has\_information\_temperature$. We set the range of $has\_information\_temperature$ to the class $Temperature$. As a result, the reasoner infers that B must be of type $Temperature$.

Inconsistency

We guard the consistency of the ontology using $DisjointClasses$. The $agrees\_with$ and $disagrees\_with$ object properties of the consensus and conflict patterns are naturally disjoint: when evidence A agrees with evidence B, they cannot disagree at the same time. The decision design pattern inherits other inconsistency prevention mechanics from the completeness, reproducibility, consensus, and conflict patterns.

Generic ontology design pattern

The decision ontology pattern instantiates the completeness, reproducibility, consensus, and conflict patterns. These patterns have an overlap in their instantiation; for example, the consensus and conflict pattern both instantiate the $Reproducibility\_Base$. We reduce the overlap by defining new instantiation code. Code sample 12 presents the instantiation code for the base ontology structure. The base instantiation code addresses the reproducibility, consensus, conflict, and a part of the completeness pattern. The base instantiation does not use any parameters.

Figure 34 presents the instantiated decision ontology pattern. This ontology does not include context.

```
pattern DecisionOntologyPattern_Basic = Reproducibility_Basic then
 %% Consensus pattern without reproducibility pattern
 ObjectProperty: agrees_with Domain: Evidence Range: Evidence Characteristics: Transitive
    , Symmetric
 ObjectProperty: based_on_evidence SuperPropertyOf: based_on_evidence o agrees_with
    SubPropertyOf: based_on_evidence
 %% Conflict pattern without reproducibility pattern
 ObjectProperty: disagrees_with Domain: Evidence Range: Evidence Characteristics:
    Symmetric, Irreflexive
 ObjectProperty: conflict_with_evidence SuperPropertyOf: based_on_evidence o
    disagrees_with SubPropertyOf: conflict_with_evidence
 %% Completeness pattern
 Completeness_dp [data_value][Information][xsd:int] %% Data property and information
    class as parameters
 Completeness_dp [data_description][Information][xsd:string] %% Data property and
    information class as parameters
 %% Used information
 Object Property: information_of InverseOf: has_information
 Class: Used_Information EquivalentTo: information_of some owl:Thing
 %% Used evidence
 Object Property: evidence_used_for InverseOf: based_on_evidence
 Class: Used_Evidence EquivalentTo: evidence_used_for some Information
```
Code sample 12: The base GDOL instantiation code for the decision ontology pattern. The instantiation code is a combination of the completeness, reproducibility, consensus, and conflict patterns.

We took the violet edges and nodes from the evidence-based management pattern, the blue edges and nodes from the reproducibility pattern, the red edges from the conflict pattern, and the green edge from the consensus pattern. The completeness pattern does not include structural elements.

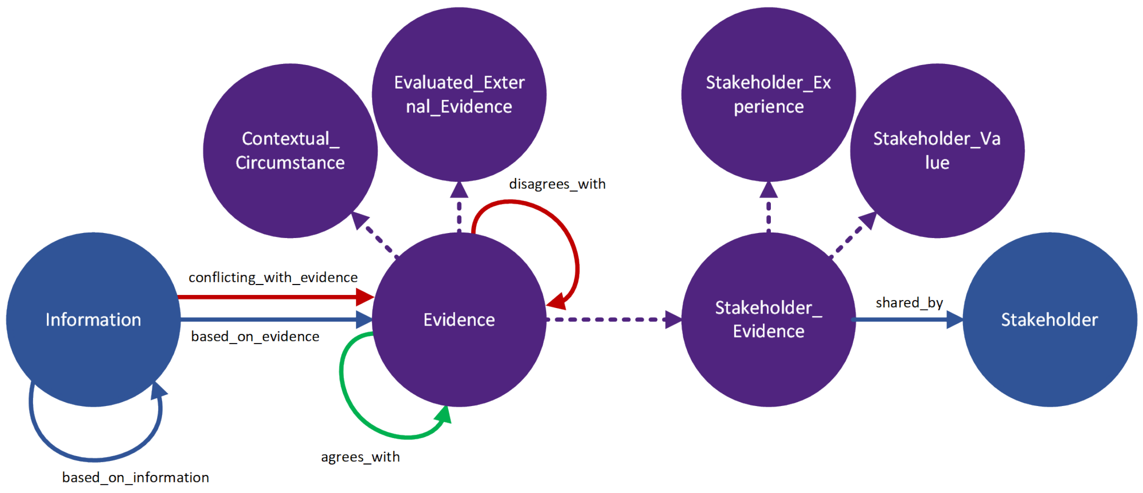


Figure 34: The instantiated decision ontology pattern without context. We took the violet edges and nodes from the evidence-based management pattern, the blue edges and nodes from the reproducibility pattern, the red edges from the conflict pattern, and the green edge from the consensus pattern.

The completeness and reproducibility patterns require a context-specific instantiation. We instantiate a new context-specific ontology structure for every information type that decision-makers use to make a decision. We use parameter i to represent this class. Code sample 13 presents the context-specific instantiation code.

Constraints

We need to combine the basic and context-dependent constraints of the completeness, reproducibility, consensus, and conflict patterns. The constraints of the reproducibility and conflict patterns target the *Used_Information* class. Code sample 14 combines the constraints for these two patterns into one

```
pattern DecisionOntologyPattern_Context [Class: i; Class: r] =
 Completeness_op[has_information[i]; r; i] %% Object property, root class, and
    information class as parameters
```

Code sample 13: The context-specific GDOL instantiation code to instantiate the decision design pattern. The code is a combination of the context-specific instantiation code of the completeness and reproducibility patterns.

SHACL shape.

```
UsedInformationShape a sh:NodeShape;
   sh:targetClass Used_Information;
   sh:property [
      sh:or (
         [sh:path based_on_information; sh:minCount 1;]
         [sh:path based_on_evidence; sh:minCount 1;]
      )
      sh:severity sh:Violation;
      sh:minCount 1;
      sh:message "Reproducibility: increase the number of evidence sources for this
    information."; ];
   sh:targetClass Information;
   # Conflict pattern
   sh:property [
      sh:path conflict_with_evidence;
      sh:severity sh:Violation;
      sh:maxCount 0;
      sh:message "Conflict detected. Please resolve the conflict or re-consider using
    this information."; ];
```

Code sample 14: This code sample combines the constraints of the reproducibility, consensus, and conflict patterns. We have merged parts of the reproducibility and consensus patterns.

Code sample 7 is part of the reproducibility pattern and presents the constraints that validate if *Stakeholder_Evidence* is *shared_by* a stakeholder. Code sample 9 presents the constraints that validate the consensus pattern. We include both code samples as-is into the decision ontology pattern.

We instantiate the contextual constraints multiple times per scenario, depending on the need of the scenario. The completeness pattern is context-sensitive. Code sample 3 presents the only context-sensitive constraints we use in the decision design pattern.

4.4 Decision presentation pattern

The outcome of a decision might deviate from the expectations of the decision-maker when the information maturity-level is low. This deviation is not a big problem when the impact of the decision on the organisation is low. However, when the impact of the decision on the organisation is high, the decision-maker wants more certainty that the decision reaches the expected outcome. There is a consensus on the benefits of evidence-based decision-making. However, Briner et al., 2009 and Baba and HakemZadeh, 2012 also raise the need for more research to prove its effectiveness.

The decision presentation pattern helps decision-makers to make evidence-based decisions by presenting the information maturity-level.

The decision presentation pattern achieves this goal by enabling the decision-maker to understand the maturity-level of the decision-relevant information. We give decision-makers two options when we present the information maturity-level:

1. Accept the status of the decision-relevant information.
2. Elaborate further on the decision-relevant information.

We enable the decision-maker to understand which information requires more elaboration. This elaboration can increase the completeness and reliability of the information. We present three dashboards for the decision-maker. The first dashboard presents the information maturity-level and evidence spread for all the decision-relevant information. The second dashboard presents the information maturity-level for the completeness, reproducibility, consensus, and conflict patterns. The third dashboard presents the maturity-levels per individual.

Figure 35 presents a navigation concept for the three dashboards. The decision-maker can browse from the first dashboard to the second dashboard, from the second dashboard to the third dashboard, and from the first dashboard to the third dashboard. The context of the third dashboard changes depending on the origin of the decision-maker. For example, if the decision-maker browses from dashboard two to dashboard three using the completeness maturity-level, dashboard three presents the completeness of the decision-relevant information per individual.

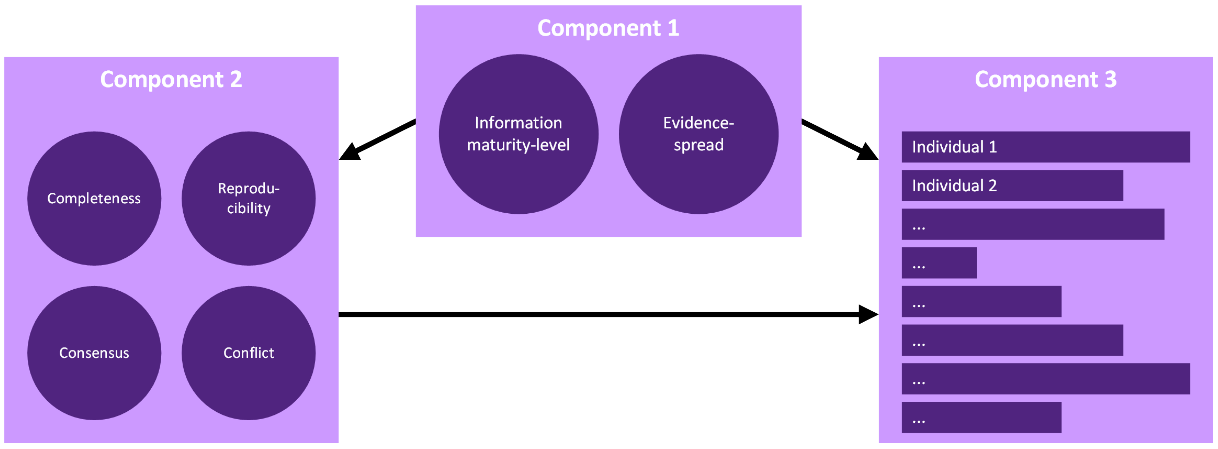


Figure 35: The three presentation dashboards that allow a decision-maker to understand the information-maturity level and evidence-spread.

4.4.1 Dashboard 1: Consolidated information maturity-level and evidence spread

The goal of the first dashboard is to help decision-makers to decide if the information maturity-level and evidence-spread are acceptable in the context of the specific decision. We determine an average information maturity-level and evidence-spread across the decision-relevant root individuals. Figure 36 repeats the definition of the root, information, and target individuals.

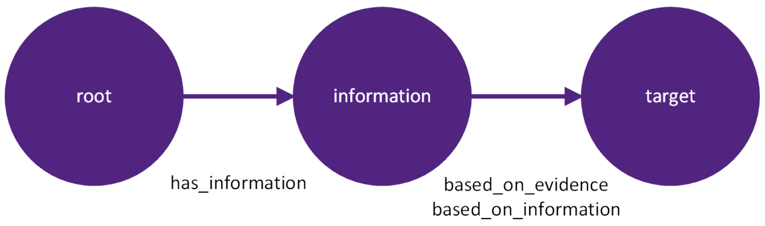


Figure 36: The *root*, *information*, and *target* individuals in n-ary relation.

We need to know the maximum number of violations before we can determine the information maturity-level. The maximum number of violations depends on the decision-relevant root individuals. We create a set RI that contains the decision-relevant root individuals. The contents of the set RI depend on the context of the decision. Therefore, we cannot define RI as part of the pattern.

Maximum number of violations: Completeness

The maximum number of violations for the completeness pattern for a specific decision-relevant root individual depends on three things:

1. The required $has\_information\_*$ object properties.
2. The availability of the $data\_value$ and $data\_description$ data properties.
3. The availability of context-specific data properties.

First, we need to retrieve the $has\_information\_*$ object properties for a specific decision-relevant root individual independent from the ontology content. We need to extract this information manually from the ontology structure by analysing the $has\_information\_*$ object properties. We represent this manual extraction process in the function $hi(ri) \Rightarrow \mathbb{N}$. ri represents a decision-relevant root individual.

Second, each $has\_information\_*$ object property leads us to an individual classified as $Information$. Information individuals require a $data\_value$ and $data\_description$. Therefore, we multiply the output of hi by 2 and add it to the result.

Third, adding additional object or data properties to the maximum number of violations for the completeness pattern might be necessary. We use variable c for adding the context-specific violations for each decision-relevant root individual.

Equation 2 returns the maximum number of violations for the completeness pattern considering the decision-relevant root individual ri as a parameter: $mvi_1(ri) \rightarrow \mathbb{N}$.

$$mvi_1(ri) = hi(ri) + 2hi(ri) + c \tag{2}$$

Equation 3 presents the maximum number of violations for the completeness pattern mv_1, considering the set of decision-relevant root individuals RI. The decision-relevant root individuals are context-dependent. We define a SPARQL query to extract the decision-relevant root individual per scenario.

$$mv_1(RI) = \sum_{ri \in RI} mvi_1(ri) \tag{3}$$

Maximum number of violations: Reproducibility

The maximum number of violations for the reproducibility pattern depends on the number of information classes that require reproduction. We use the same concept to retrieve the maximum number of violations for the reproducibility as we did for the completeness pattern.

Equation 4 returns the maximum number of violations for the reproducibility pattern considering the decision-relevant root individual ri as a parameter: $mvi_2(ri) \rightarrow \mathbb{N}$.

$$mvi_2(ri) = hi(ri) \tag{4}$$

Equation 5 presents the maximum number of violations mv_2 for the reproducibility pattern considering the set of decision-relevant root individuals RI.

$$mv_2(RI) = \sum_{ri \in RI} mvi_2(ri) \tag{5}$$

Maximum number of violations: Consensus

The maximum number of violations for the consensus pattern depends on the maximum number of used evidence sources in the context of a decision. Each evidence source can generate one violation.

Code sample 15 presents a SPARQL query that counts the $Used\_Evidence$ individuals based on the object property $evidence\_used\_for$. $evidence\_used\_for$ is the inverse of $based\_on\_evidence$.

```
SELECT (COUNT(DISTINCT ?t) as ?count)
WHERE
{
   ?r has_information ?i .
   ?i based_on_evidence ?t .
   ?t rdf:type Used_Evidence .
   FILTER (?r = <ri_1>)
}
```

Code sample 15: The SPARQL query that retrieves the number of individuals that the reasoner classified as $Used\_Evidence$ in the context of a specific root class.

Equation 4 returns the maximum number of violations for the reproducibility pattern considering the decision-relevant root individual ri as a parameter: $mvi_3(ri) \rightarrow \mathbb{N}$. Function $uei(ri) \rightarrow \mathbb{N}$ represents the output of the SPARQL query code sample 15 presents.

$$mvi_3(ri) = uei(ri) \tag{6}$$

We cannot sum the results of the individual violations as there might be duplicates in the results. For example, two information individuals might use the same evidence. When counting the maximum number of violations per individual, the sum would count this evidence as two violations. However, considering one decision, one evidence source can generate up to one violation. Code sample 16 presents a new filter that we apply on code sample 15. We add the entire content of the set RI to the filter. This way, we ensure that we get a list of evidence sources that is distinct for the specific decision.

```
FILTER (?r = <ri_1> || ?r = <ri_2> || ?r = <ri_x>)
```

Code sample 16: A new filter that we apply on code sample 15. We add the entire content of the set RI to the filter.

Equation 7 presents the maximum number of violations mv_3 for the reproducibility pattern considering the set of decision-relevant root individuals RI. Function $ue(RI) \rightarrow \mathbb{N}$ represents the SPARQL query code sample 16 presents.

$$mv_3(RI) = ue(RI) \tag{7}$$

Maximum number of violations: Conflict

The conflict pattern detects information individuals that have conflicting evidence using the $conflict\_with\_evidence$ object property. The pattern does not care about the number of conflicting evidence related to an individual. Alternatively, we could have used a similar mechanism as we used for the consensus pattern and detect conflict based on the $disagrees\_with$ object property. However, we solve the violation in the consensus pattern by adding an $agrees\_with$ object property to the evidence. The decision-maker can solve the conflict detected by the pattern by removing conflicting evidence or changing the information. In other words, the consensus pattern uses the $Evidence$ class as the core of the solution, while the conflict pattern uses the $Information$ class as the core of the solution.

The maximum number of conflict violations for a decision-relevant root individual depends on the maximum number of information classes that can be evidence-based. We re-use equation mvi_2 to determine the maximum number of conflict violations. Equation 8 presents the maximum number of violations mvi_4 for the conflict pattern.

$$mvi_2(ri) = mvi_4(ri) = hi(ri) \tag{8}$$

Equation 9 presents the maximum number of violations mv_4 for the reproducibility pattern considering the set of decision-relevant root individuals RI. mv_4 is equal to mv_2.

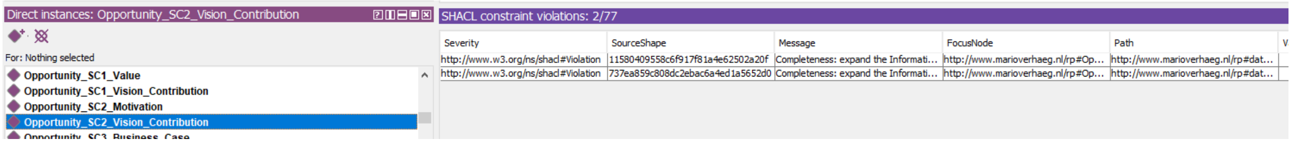
$$mv_4(RI) = \sum_{ri \in RI} mvi_4(ri) \tag{9}$$

Consolidated information maturity-level

We calculate the information maturity-level using the number of violations detected on the decision-relevant information and the maximum violations. For example, if an individual requires ten object properties and two data properties to be complete, and it triggers two violations, we define the information maturity-level as $(12-2)/12 = 83\%$. Equation 10 includes function $mv(RI) \Rightarrow \mathbb{N}$. This function calculates the maximum number of violations for the set of decision-relevant information. $av(RI) \Rightarrow \mathbb{N}$ calculates the actual number of violations for the set of decision-relevant information and $iml(RI) \Rightarrow \mathbb{N}$ represents the information maturity-level for the set.

$$iml(RI) = \frac{mv(RI) - av(RI)}{mv(RI)} \tag{10}$$

We can easily retrieve the actual number of violations from, for example, Protégé and the SHACL4P plugin. Figure 37 presents an example for which the defined constraints detected two violations for *Opportunity_SC2_Vision_Contribution*.


Direct instances: Opportunity_SC2_Vision_Contribution

For: Nothing selected

- Opportunity_SC1_Value
- Opportunity_SC1_Vision_Contribution
- Opportunity_SC2_Motivation
- Opportunity_SC2_Vision_Contribution
- Opportunity_SC3_Business_Case

SHACL constraint violations: 2/77

Severity	SourceShape	Message	FocusNode	Path	V
http://www.w3.org/ns/shacl#Violation	11580409558c6f917f81a4e62502a20f	Completeness: expand the Informati…	http://www.marioverhaeg.nl/rp#Op…	http://www.marioverhaeg.nl/rp#dat…	
http://www.w3.org/ns/shacl#Violation	737ea859c808dc2ebac6a4ed1a5652d0	Completeness: expand the Informati…	http://www.marioverhaeg.nl/rp#Op…	http://www.marioverhaeg.nl/rp#dat…	

Figure 37: The constraints detect two violations for *Opportunity_SC2_Vision_Contribution* in Protégé.

Additionally, we need to define the maximum number of violations per decision-relevant root individual and pattern. The maximum number of violations per pattern for a specific individual defines the total maximum number of violations: $mv(RI) = mv_1(RI) + mv_2(RI) + mv_3(RI) + mv_4(RI)$, in which:

1. $mv_1(RI)$ is the maximum number of violations for the completeness pattern.
2. $mv_2(RI)$ is the maximum number of violations for the reproducibility pattern.
3. $mv_3(RI)$ is the maximum number of violations for the consensus pattern.
4. $mv_4(RI)$ is the maximum number of violations for the conflict pattern

RI represents the set of decision-relevant root individuals.

Evidence spread

We increase the transparency of the information maturity-level by presenting the evidence spread. For example, a decision entirely based on the evidence type *Stakeholder_Experience* might require more elaboration. We use a query to extract the evidence spread from the decision ontology pattern. Within this query, we search for root individuals ri that represent decision-relevant information. The filter in the query accepts multiple root individuals using || the (OR) filter criteria. Code sample 17 presents the SPARQL query.

```
SELECT ?t (COUNT(DISTINCT ?t) as ?count)
WHERE
{
   {
         ?ri has_information ?i .
         ?i based_on_evidence ?e .
         ?e rdf:type ?t .
   }
   FILTER(?ri = <ri1> || ?ri = <ri2> || ?rix = <rix> ).
   FILTER(?t != owl:Thing && ?t != Evidence && ?t != Used_Evidence && ?t !=
    Stakeholder_Evidence)
}
GROUP BY ?t
```

Code sample 17: The SPARQL query that retrieves the evidence spread for decision-relevant information. The SPARQL query counts the individuals that are based on a specific evidence class using type $?t$ of evidence class $?e$.

Overview dashboard

The goal of the first high-level dashboard is to help the decision-maker answering the question:

Is the information ready for an evidence-based decision?

This dashboard consists of two charts: the overall information maturity-level and the evidence spread. The information maturity-level helps the decision-maker to understand the quality-level of the information related to this decision. The evidence-spread helps the decision-maker to understand the source of the evidence. The decision-maker needs to find a balance between the information maturity-level and the decision impact, and the evidence-spread and the decision impact. When the impact of a decision is relatively low, the decision-maker might accept a lower information maturity-level and a consolidated evidence-spread. However, when the impact of a decision is relatively high, we expect that decision-makers will require a higher information maturity-level and would like to see a dispersed evidence-spread.

We use a pie-chart to help the decision-maker to understand the mix of evidence-types. The pie-chart is suitable to present relative proportions and percentages (Hardin et al., 2012).

We express the information maturity-level as a percentage. Therefore, we use a pie-chart for presenting the information maturity-level as well. Figure 38 presents an example of a dashboard that contains the pie chart presenting the evidence-spread and the information maturity-level. We shorten the time it takes for a decision-maker to consume the information on the dashboard by limiting the number of charts, using data labels to show the actual values, and using a clear legend (Coles, 1997). Additionally, we use an analogous colour scheme for the evidence spread to ensure the chart appears as one, but the colours are still easily recognisable (Stone, 2006). The pie chart presenting the information maturity-level uses a different colour scheme, representing *good* (the mature information) and *bad* (the premature information) as green and red.

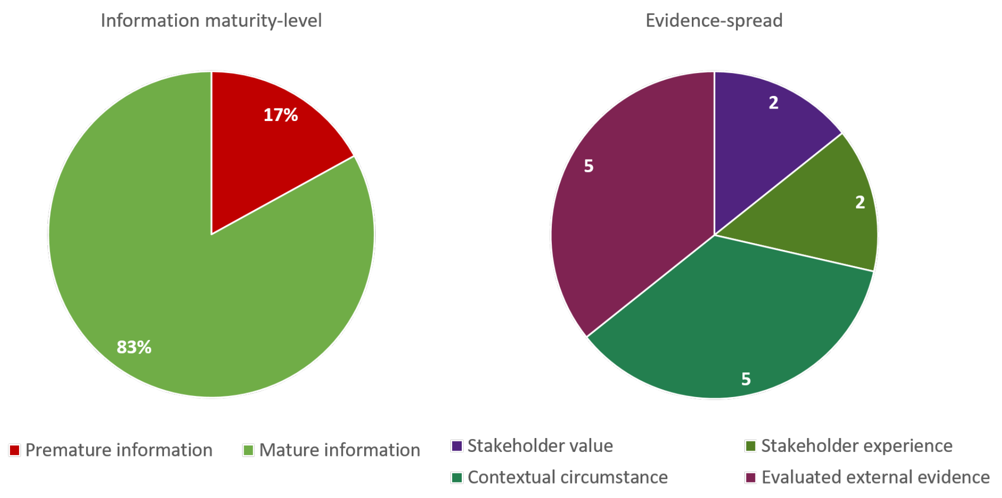


Figure 38: A mock-up of the presentation of the evidence-mix of a specific decision.

The decision-maker can navigate to the second dashboard by clicking on the pie chart that presents the information maturity-level. Alternatively, the decision-maker can navigate to the third dashboard by clicking on the pie chart that presents the evidence-spread.

4.4.2 Dashboard 2: Information maturity-level in detail

The goal of the second dashboard is to enable a decision-maker to understand the consolidated information maturity-level. If the consolidated information maturity-level is lower than expected, the decision-maker can use the second dashboard to understand the root cause. This understanding allows the decision-maker to define actions to improve the information maturity-level. For example, the decision-maker needs to spend time on expanding the information if the completeness pattern causes a low information maturity-level. However, the decision-maker should understand the conflicts and reconsider using conflicting evidence sources if these evidence sources cause a low information maturity-level.

Pattern information maturity-levels

The detailed information maturity-level consist out of the maturity-level for the completeness, reproducibility, consensus, and conflict patterns. We have defined the functions mv_1, mv_2, mv_3, and mv_4 for this.

In addition to the actual violations av, we define $av_1(RI) \Rightarrow \mathbb{N}$, $av_2(RI) \Rightarrow \mathbb{N}$, $av_3(RI) \Rightarrow \mathbb{N}$, and $av_4(RI) \Rightarrow \mathbb{N}$ to represent the actual violations for the completeness, reproducibility, consensus, and conflict patterns. We also define the $avi_x(ri)$ functions that represent the actual violations that a decision-relevant individual generates. Equation 11 presents the information maturity-level of the decision-relevant root individuals RI for one of the patterns.

$$iml_x(RI) = \frac{mv_x(RI) - av_x(RI)}{mv_x(RI)} \tag{11}$$

Detailed information maturity-level dashboard

Equation 11 defines four pattern-specific information maturity-levels. The pie-chart is most suitable to present percentages and relative proportions (Hardin et al., 2012). Figure 39 presents an example of a dashboard that contains the pie charts that present the pattern-specific information maturity-levels. We have made the pie charts easy to understand by tailoring the scales to the specific pattern, representing *good* (the mature information) and *bad* (the premature information) as green and red, and using data labels. The decision-maker can navigate to the third dashboard by clicking on one of the pie charts.

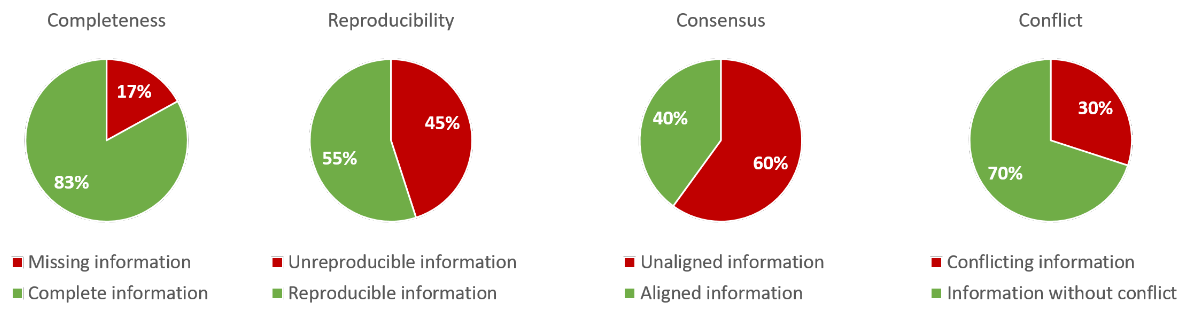


Figure 39: A mock-up of the presentation of the pattern-specific information maturity-levels.

4.4.3 Dashboard 3: Violations

The goal of the third dashboard is to enable the decision-maker to understand which individuals cause the lowered information maturity-level and the evidence spread. The decision-maker might have the following questions:

1. How can I diversify the evidence types to increase the information maturity-level?
2. Which information is missing?
3. Which information is unreproducible or does not meet the consensus-level?
4. Which information has evidence conflicts?

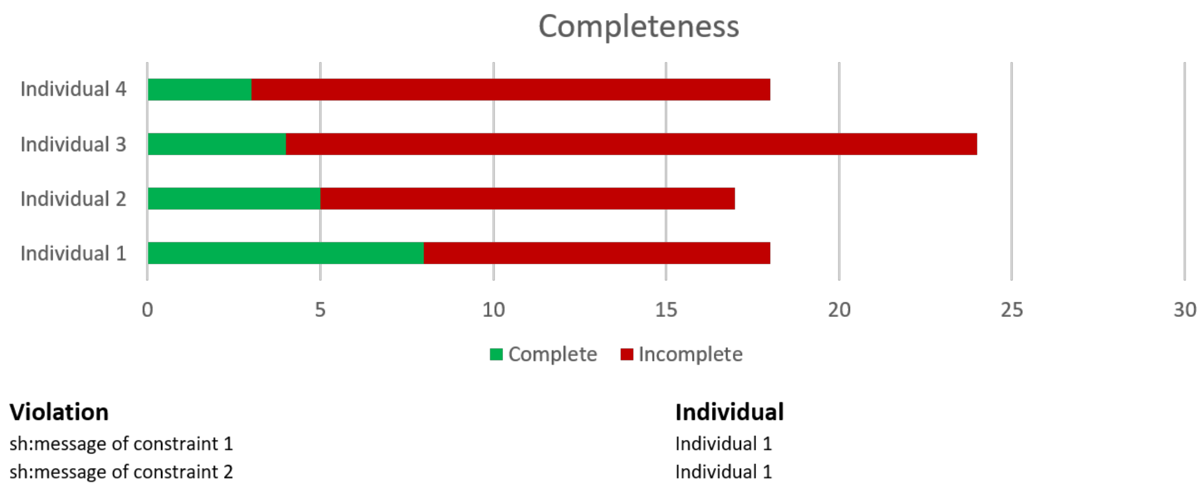
The answer to these questions allows the decision-maker to increase the information maturity-level. The decision-maker can complete information by entering new information, can reproduce information by linking it to existing or new evidence sources, or remove conflict by discarding information or replacing evidence sources. We focus on the data structure, detection mechanisms, and data presentation. Creating forms to enter new information or change existing information is, therefore, not in the scope of this study.

We tailor the presentation of the third dashboard, depending on these four questions. The structure of the presented information is similar for the four use-cases. The level of detail requires us to present the information per individual. As a result, we need to present multiple individuals in the same environment. We only present premature individuals to prevent overwhelming the decision-maker with information. Presenting the mature individuals does not make it easier for the decision-maker to answer the questions mentioned earlier.

We want the decision-maker to see the difference between the categories of information in each of the four questions, for example, complete versus incomplete. Presenting the diversity of evidence types for multiple individuals requires four categories: evaluated external evidence, stakeholder value, stakeholder experience, and contextual circumstances. A bar chart allows the decision-maker to compare data across categories (Hardin et al., 2012).

Individual dashboard

Figure 40 presents an example of the presentation of the completeness maturity-level of four individuals. We show the completeness maturity-level for each individual using a bar chart. The number of individuals depends on the scope of the decision. We present the related violations in a table below the bar chart for the selected decision. Even though entering or changing information is not in the scope of this study, we can easily imagine that clicking on one of the violations would direct the decision-maker to a form that allows the decision-maker to enter or adjust information and, as a result, solve the violation immediately. We use the same presentation for the reproducibility, consensus, and conflict-related questions. However, we replace the header of the chart and the name of the data series.



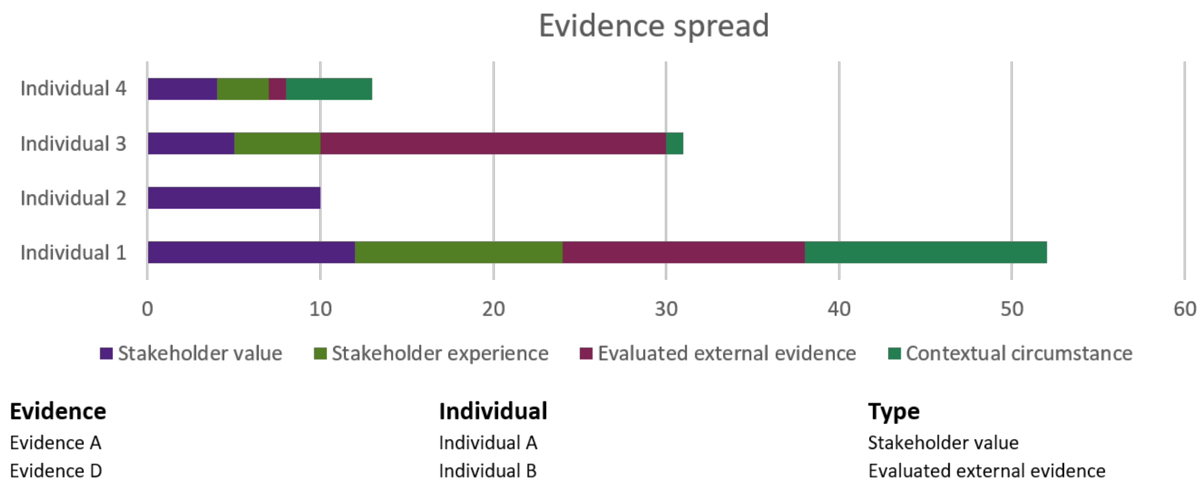
Violation	Individual
sh:message of constraint 1	Individual 1
sh:message of constraint 2	Individual 1

Figure 40: A mock-up of the presentation of the completeness maturity-level of four individuals. We show the completeness maturity-level for each individual using a bar chart.

We extract the actual individual violations from the ontology. The mvi_1, mvi_2, mvi_3, and mvi_4 functions define the maximum number of violations per individual. We subtract the actual violations av from the maximum violations mv to get the information without conflict, the complete information, the reproducible information, and the aligned information.

Evidence spread dashboard

The presentation of the evidence spread is a bit different. Instead of two data series, we need four data series: one for each evidence type. Figure 41 presents an example of the presentation of the evidence spread of four individuals. We have chosen for a consistent colouring scheme, considering figure 38. We present the evidence spread for each individual using a bar chart. There are no constraints that enforce a specific evidence spread. Therefore, there are no violations to present. Instead, we present a list of evidence related to the selected individual below the bar chart. Furthermore, we consolidate the list of violations based on the root individuals. This presentation allows a decision-maker to understand the evidence spread and actual evidence related to a root individual.



Evidence	Individual	Type
Evidence A	Individual A	Stakeholder value
Evidence D	Individual B	Evaluated external evidence

Figure 41: A mock-up example of the presentation of the evidence spread of four individuals. We have chosen for a consistent colouring scheme, considering figure 38. We present the evidence spread for each individual using a bar chart.

We considered using an alternative presentation based on graphs. Figure 42 presents an example of this presentation. The graph-based presentation overwhelms the decision-maker with a lot of details, including the relationships between decision-relevant information and their status. This example presents only seven decision-relevant individuals. A more extensive example increases the size and complexity of the presentation. Additionally, the edges do not add useful information for the decision-maker.

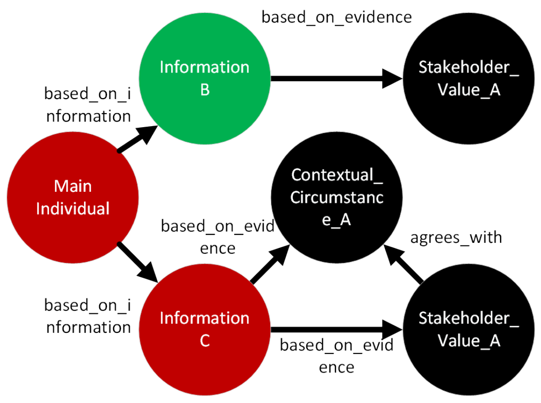


Figure 42: An alternative, more complex, presentation of the decision-relevant individuals and their maturity status.

5 Validation

The decision ontology pattern consolidates the completeness, reproducibility, consensus, and conflict patterns. These patterns are useful on their own. However, only their consolidation, combined with the evidence-management pattern, contributes to the answer to our research question. Therefore, we validate the consolidated decision ontology pattern in this chapter. Additionally, we also validate the decision presentation pattern in this chapter.

We use two scenarios to validate the decision ontology pattern and decision presentation pattern:

1. Scenario 1: Requirements prioritisation
2. Scenario 2: Alternative solution selection

The goal of the first scenario is to prove the decision design pattern leads to the expected results. The goal of the second scenario is to prove that the decision design pattern can be applied in other scenarios as well. We limit the validation depth of the second scenario when there is an overlap with the first scenario.

These scenarios are related to each other. A software product manager first prioritises the requirements in his backlog. Each requirement specifies a problem to solve or a goal to reach. Once the product team picks a requirement for implementation, the product team uses alternative solution selection to select the right technical solution to solve the specified problem or reach a specific goal. Both scenarios have challenges that attract the interest of academics. We use these scenarios as examples for which the decision design pattern increases the transparency of the information maturity-level. Software product managers can use this transparency to evaluate if the information is ready to make an evidence-based decision.

5.1 Scenario 1: Requirements prioritisation

The goal of the requirements prioritisation scenario is to validate if the decision design pattern can help software product managers to use evidence-based requirements prioritisation.

$VAL1$: To what extent can the detection of premature information using SHACL Semantic Web constraints contribute to evidence-based requirements prioritisation?

A software product manager needs to decide if one requirement is more important than another requirement. The software product manager contributes to reaching the defined goals of the product when the software product manager puts the requirement on the right position in the backlog. These goals are, for example, related to the revenue of the product or the size of its user base. When the software product manager puts the requirement on a higher position in the backlog, then it is supposed to be, another requirement is on a lower position then it should be.

5.1.1 Decision ontology pattern

We base the definition of the decision ontology pattern for requirements prioritisation on section 2.5.2 Requirements prioritisation of the theoretical framework. Table 4 presents the knowledge criteria that a software product manager uses to decide which requirement, out of two, is most important. We define the $Requirement$ and $Insight$ (with sub-classes $Challenge$ and $Opportunity$) as decision-relevant *root* classes. We need the $Vision$ class for the reproduction of information. Figure 43 presents the ontology.

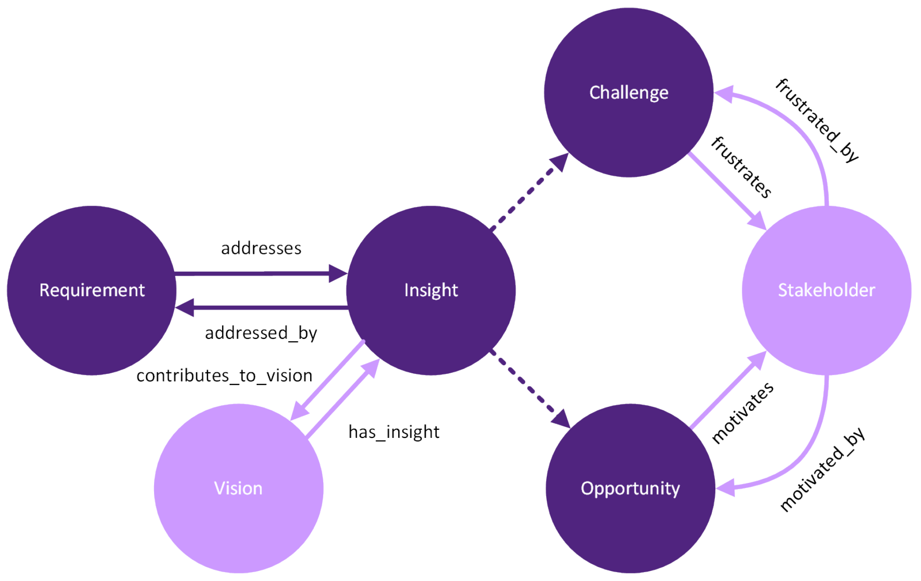


Figure 43: The requirements prioritisation ontology. The violet nodes represent the decision-relevant root classes. The light violet nodes represent the classes we need for the reproduction pattern.

The *addresses* and *contribute_to_vision* object properties are essential for the structural integrity of the ontology. A *Requirement* that does not address an *Insight* and an *Insight* that does not contribute to a *Vision* are structural problems. Missing these object properties influences the reliability of the information that we present in the dashboards.

5.1.2 Instantiation of the decision ontology pattern

We instantiate the base decision ontology pattern once for the requirement prioritisation scenario. Code sample 18 presents the instantiation of the base decision ontology pattern. We expand the decision ontology pattern with the ontology figure 43 presents manually.

```
Ontology RequirementsPrioritisation = DecisionOntologyPattern_Basic
```

Code sample 18: The GDOL instantiation code of the basic instantiation of the decision design pattern.

The requirements prioritisation ontology and the instantiated decision design pattern include a *Stakeholder* class. We manually merge the *Stakeholder* classes from these ontologies into a single *Stakeholder* class. Alternatively, we could have decided to define both that *Stakeholder* classes are equal. However, merging the two *Stakeholder* classes into one class reduces the number of classes and, therefore, reduces the complexity of the ontology.

The context-specific instantiation of the decision design pattern requires parameters to extend the ontology with a context-specific structure. We use the context-specific instantiation to, for example, instantiate the knowledge criteria in a way we can validate their completeness and reproducibility.

Table 4 presents the knowledge criteria that a software product manager needs to understand to decide if one requirement is more important than another requirement. We add the knowledge criteria to the domain of the *Insight* and *Requirement* classes of the requirements prioritisation ontology. We define the completeness-level of a *Requirement* and *Insight* based on the knowledge criteria.

Table 4: An overview of the formal knowledge criteria that requirement prioritisation needs based on the description in section 2.5.2 Requirements prioritisation. Code sample 19 instantiates the knowledge criteria.

Data Property	Domain
insight_value	*Addressed_Insight*
insight_vision_contribution	*Addressed_Insight*
insight_reach	*Addressed_Insight*
requirement_insight_contribution	*Requirement*
requirement_value	*Requirement*
requirement_cost	*Requirement*
requirement_confidence	*Requirement*

We instantiate each knowledge criteria using the generic ontology design pattern presented by code sample 13. The generic ontology design pattern *DecisionDesignPattern_Context* uses two parameters: the information class and the decision-relevant root class. Code sample 19 presents, for example, the instantiation of the *insight_value* as an example. Code sample 20 shows the instantiated generic ontology design pattern.

```
DecisionOntologyPattern_Context[insight_value][Addressed_Insight]
```

Code sample 19: The GDOL instantiation code of the knowledge criteria.

```
Completeness_op [has_information_insight_value][Addressed_Insight][insight_value]
```

Code sample 20: The result of the GDOL instantiation code that code sample 19 presents.

Code sample 21 shows the instantiated SHACL shapes for, for example, the *insight_value*. We need to adjust the $sh : targetClass$, $sh : path$, and $sh : message$ variables manually.

The constraints generate three violations when a knowledge criterion is not available. One violation represents the unavailability of the knowledge criterion itself, and the other two violations represent the missing *data_value* and *data_description*. It is possible to add more violations if the context requires this.

The reproduction of knowledge criteria uses the completeness pattern. With the instantiation of the generic ontology design pattern and the constraints, we can validate the reproducibility of the knowledge criteria throughout the information chain figure 44 shows.

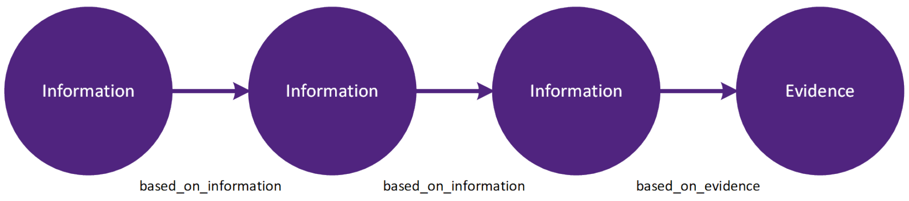


Figure 44: An example of an evidence-based chain of information. The reproducibility pattern should detect if the chain is not evidence-based.

```
rp:Addressed_InsightShape a sh:NodeShape;
   sh:targetClass rp:Addressed_Insight;
   sh:property [
      sh:path rp:has_information_insight_value;
      sh:severity sh:Violation;
      sh:minCount 1;
      sh:message "Completeness: add insight_value to the Addressed_Insight."; ];
   sh:property [
      sh:path rp:has_information_insight_value;
      sh:severity sh:Violation;
      sh:minCount 1;
      sh:message "Completeness: add data_value to the insight_value."; ];
   sh:property [
      sh:path rp:has_information_insight_value;
      sh:severity sh:Violation;
      sh:minCount 1;
      sh:message "Completeness: add data_description to the insight_value."; ];
```

Code sample 21: An example of instantiated SHACL shapes for the data property *insight_value*. We instantiate this code sample for each knowledge criterion and for each information required to reproduce a knowledge criterion.

Insight value

The insight value takes input from the stakeholder on the *frustration* (challenge) or *motivation* (opportunity) the stakeholder experiences. We consider a description of the *business case* from the stakeholder's perspective as well. Table 5 presents an overview of the background information the software product manager requires to define the insight value.

Table 5: The background information that we require to reproduce the value of an insight.

*has_information_**	Class	Description
challenge_ frustration	*Addressed_Challenge*	The current pain the stakeholder experiences. A stakeholder finds this insight a necessary *fix*.
opportunity_ motivation	*Addressed_Opportunity*	The motivation of a stakeholder to use the product more often if a requirement addresses this insight.
insight_ business_case	*Addressed_Insight*	Increased revenue, decreased cost, or decreased business risk.

Code sample 22 presents the instantiation of the *Insight_Value*.

```
DecisionDesignPattern_Context[challenge_frustration; Addressed_Challenge]
DecisionDesignPattern_Context[opportunity_motivation; Addressed_Opportunity]
DecisionDesignPattern_Context[insight_business_case; Addressed_Insight]
```

Code sample 22: The GDOL instantiation code of the information reproducing the *Insight_Value*

Vision contribution

The vision itself needs to be defined to validate if the insight contributes to the vision. We introduce the *Vision* class for this. The vision class hosts the vision statement (and related time frame), a measurable objective (and related time frame), a current state and target condition. Table 6 presents an overview of the background information the software product manager requires to define the vision contribution.

Table 6: The background information that we require to reproduce the vision contribution of an insight.

*has_information_**	Class	Description
vision_ statement	*Used_Vision*	A futuristic goal defined in the context of a product or company.
vision_measureable_objective	*Used_Vision*	A description of a measurable objective that achieves the first step towards a futuristic goal.
vision_target_condition	*Used_Vision*	The first smaller objective that contributes to achieving a measurable objective.
vision_current_state	*Used_Vision*	A description of today's status related to the target condition.

We can reproduce the *Insight_Vision_Contribution* based on data properties stored in an *Insight* and *Vision*. The domain of the *Insight_Vision_Contribution* itself is an *Insight*.

Code sample 23 presents the instantiation of the *Insight_Vision_Contribution*.

```
DecisionDesignPattern_Context[vision_statement; Used_Vision]
DecisionDesignPattern_Context[vision_measurable_objective; Used_Vision]
DecisionDesignPattern_Context[vision_target_condition; Used_Vision]
DecisionDesignPattern_Context[vision_current_state; Used_Vision]
```

Code sample 23: The GDOL instantiation code of the information reproducing the *Insight_Vision_Contribution*

Additionally, code sample 24 presents the instantiation code to validate that the time frame of the vision statement and the time frame of the measurable objective are complete. Validating the completeness of the time frames is outside of the scope of the decision design pattern as these are very context-specific. Therefore, we use the instantiation of the completeness pattern directly.

```
Completeness_dp[time_frame; vision_measurable_objective; xsd:dateTime]
Completeness_dp[time_frame; vision_statement; xsd:dateTime]
```

Code sample 24: The GDOL instantiation code of the time frames of the *vision_measurable_objective* and the *vision_statement*.

The default constraints cover the missing information, the related *data_value*, and *data_description*. Missing decision-relevant information typically results in three violations. In this case, we need to add the *time_frame* to the *vision_measurable_objective* and the *vision_statement* and generate four violations. Therefore, we add two additional constraints to the *VisionShape*. Code sample 25 presents these two additional constraints.

Insight reach

Figure 43 presents an ontology that allows multiple stakeholders to raise their frustration or motivation towards a specific insight. Ideally, each stakeholder is represented in the ontology as an individual and connected to an *Insight* using the *motivates* or *frustrates* object properties. In this case, we can easily calculate the *insight_reach* by taking the percentage of stakeholders, out of the total pool of stakeholders that are frustrated or motivated by the insight. Table 7 presents an overview of the background information the software product manager requires to define the insight reach.

```
sh:property [
   sh:path rp:has_information_vision_statement;
   sh:severity sh:Violation;
   sh:minCount 1;
   sh:message "Completeness: add time_frame to the vision_statement."; ];
sh:property [
   sh:path rp:has_information_vision_measurable_objective;
   sh:severity sh:Violation;
   sh:minCount 1;
   sh:message "Completeness: add time_frame to the vision_measurable_objective."; ];
```

Code sample 25: The SHACL shapes that validate the completeness of the *time_frame* data property in the context of the vision statement and the measurable objective.

Table 7: The background information that we require to reproduce the reach of an insight.

Object property	Domain	Range
motivates	*Opportunity*	*Stakeholder*
frustrates	*Challenge*	*Stakeholder*

Code sample 26 presents the instantiation of the *Insight_Vision_Contribution*. In this case, we create a direct link between the requirements prioritisation ontology and the reproducibility pattern. Therefore, we use the instantiation of the completeness pattern. However, it is nearly impossible to get all of the stakeholders registered and capture their specific insights. Alternatively, we base the *insight_reach* on, for example, evaluated external evidence. In this case, the *insight_reach* should be reproducible by evidence.

```
Completeness_op[motivates; Addressed_Opportunity; Stakeholder]
Completeness_op[frustrates; Addressed_Challenge; Stakeholder]
```

Code sample 26: The instantiation of the completeness pattern that contributes to the reproducibility of the *insight_value*.

Requirement cost

The cost of the requirement depends on the time it takes to implement the requirement, the cost of purchasing equipment[4], and the cost of gaining knowledge that is required to realise the requirement. The team that defines this information implements the requirement as well. Table 8 presents an overview of the background information the software product manager requires to define the requirement cost.

[4]The cost of purchasing equipment can be based on, for example, a quotation classified as validated external evidence.

Table 8: The background information that we require to reproduce the cost of a requirement.

*has_information_**	Class	Description
requirement_ knowledge_cost	*Requirement*	The estimated total cost to acquire knowledge that allows the team to implement the requirement.
requirement_ equipment_cost	*Requirement*	The estimated total purchasing costs of the equipment needed to implement the requirement. The cost can include, for example, processing hardware, sensors, and software licenses.
requirement_size	*Requirement*	The estimated size of the requirement. The estimated size is a relative value, and its range depends on the way of working of the team.

Code sample 27 presents the instantiation of the *Requirement_Cost*.

```
DecisionDesignPattern_Context[requirement_knowledge_cost; Requirement]
DecisionDesignPattern_Context[requirement_equipement_cost; Requirement]
DecisionDesignPattern_Context[requirement_size; Requirement]
```

Code sample 27: The GDOL instantiation code of the information reproducing the *Requirement_Cost*

Requirement value

The *insight_value* represents the value of the insight for the stakeholder. The *requirement_value* represents the value of the requirement for the organisation developing the requirement. The requirement value typically ranges from an opportunity to increase revenue, decrease costs, or decrease certain risks. Table 9 presents an overview of the background information the software product manager requires to define the value of the requirement.

Table 9: The background information that we require to reproduce the value of a requirement.

*has_information_**	Class	Description
requirement_ increase_revenue	*Requirement*	The description of how this requirement increases the revenue of the organisation developing the requirement.
requirement_ decrease_cost	*Requirement*	The description of how this requirement decreases the costs (or increases the efficiency) of the organisation developing the requirement.
requirement_ decrease_risk	*Requirement*	The description of how this requirement decreases a particular risk the organisation faces.

Code sample 28 presents the instantiation of the *Requirement_Cost*.

```
DecisionDesignPattern_Context[requirement_increase_revenue; Requirement]
DecisionDesignPattern_Context[requirement_decrease_cost; Requirement]
DecisionDesignPattern_Context[requirement_decrease_risk; Requirement]
```

Code sample 28: The GDOL instantiation code of the information reproducing the *Requirement_Value*

Insight contribution

A product manager can address one insight using different requirements. The team that addresses the insight can slice the insight into various requirements. The requirement contribution to an opportunity depends on the motivation we can spark with the requirement. Alternatively, the *requirement_contribution* to the challenge depends on the frustration we can take away using the requirement. Table 10 presents an overview of the background information the software product manager requires to define the insight contribution.

Table 10: The background information that we require to reproduce the insight contribution of a requirement.

*has_information_**	Class	Description
challenge_frustration	*Addressed_Challenge*	The frustration we can take away by implementing the requirement.
opportunity_motivation	*Addressed_Opportunity*	The motivation we can spark by implementing the requirement.

Code sample 29 presents the instantiation of the *Requirement_Insight_Contribution*. We ensure that challenges that are addressed by an insight have a challenge, and opportunities that are addressed by a requirement have a motivation.

```
DecisionDesignPattern_Context[challenge_frustration; Addressed_Challenge]
DecisionDesignPattern_Context[motivation_opportunity; Addressed_Opportunity]
```

Code sample 29: The GDOL instantiation code of the information reproducing the *Requirement_Insight_Contribution*

Requirement confidence

The requirement confidence represents the confidence of the team that implements the requirement. The team is aware that a larger requirement is more time consuming to implement than a smaller requirement. Teams that need to acquire a lot of knowledge might be less confident compared to teams that already have most of the knowledge to realise the requirement. Table 11 presents an overview of the background information the software product manager requires to define the requirement confidence. We instantiated the *requirement_knowledge_cost* and *requirement_size* in the context of the *requirement_cost*.

Table 11: The background information that we require to reproduce the cost of a requirement.

*has_information_**	Class	Description
requirement_knowledge_cost	*Requirement*	The estimated total cost to acquire knowledge that allows the team to realise the requirement.
requirement_size	*Requirement*	The estimated size is a relative value, and its range depends on the way of working of the team.

Instantiation overview

We combine the requirements prioritisation ontology and the decision ontology pattern. Additionally, we instantiate a new class for each information type, for example, the *Requirement_Size*. The new information types are sub-classes of the *Information* root classes. The decision ontology pattern validates that the individuals classified as *Information* and the information types that are subclasses of *Information* are evidence-based. Figure 45 presents a conceptual overview of the instantiation.

We have summarized the information types in three main information classes: $Vision\_*$ includes the information classes related to the vision, $Requirement\_*$ includes the information classes related to the requirement, and $Insight\_*$ includes the information classes related to the insight.

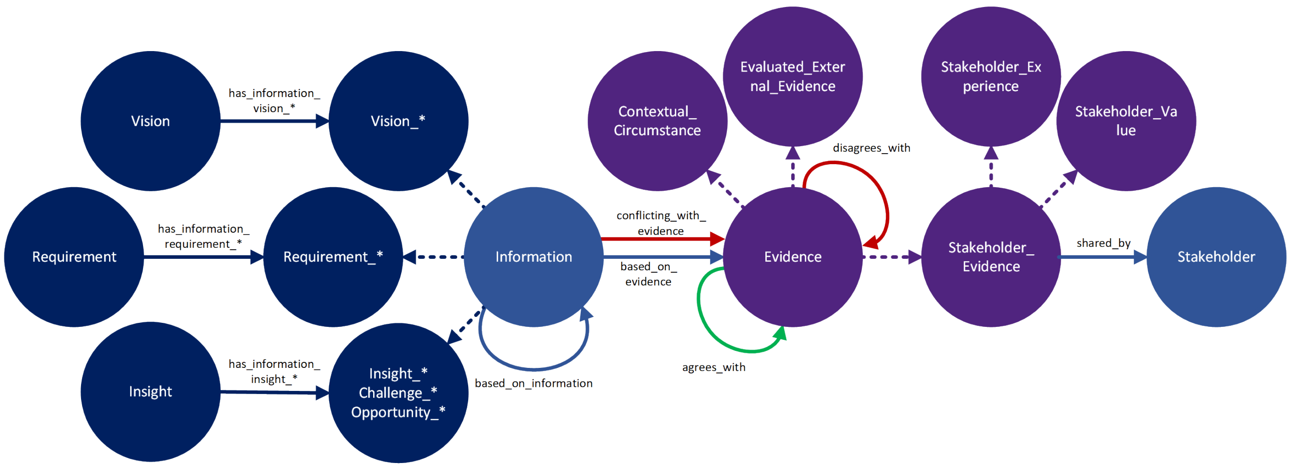


Figure 45: The combined requirements prioritisation ontology and the decision design pattern ontology.

5.1.3 Structural validation

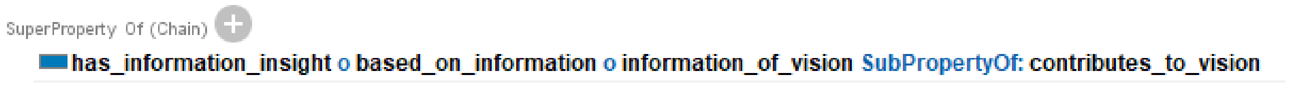
An incorrect maximum number of violations can cause an incorrect information maturity-level. The structural violations are not part of the functions that define the information maturity-level. Two scenarios can cause an incorrect maximum number of violations:

1. A requirement does not address an insight. In this case, we cannot retrieve the maximum number of violations for the insight.
2. An insight does not contribute to a vision. In this case, we cannot retrieve the maximum number of violations for the vision.

While the reasoner finds structural violations, the dashboards are not reliable and will not show up. The decision-maker needs to solve the structural violations first.

Structural validation: $Insight$ **contributes to** $Vision$

We add the object properties $has\_insight$, $contributes\_to\_vision$, and $information\_of$ (including the sub-object properties $information\_of\_requirement$, $information\_of\_vision$, and $information\_of\_insight$) to detect the visions that have an $Insight$. The $information\_of$ object property (and its sub-object properties) are the inverse of the $has\_information$ object property (and its sub-object properties). The reasoner infers the $contributes\_to\_vision$ object property from the super property figure 46 presents. $has\_insight$ is the inverse of $contributes\_to\_vision$.



SuperProperty Of (Chain)

has_information_insight o based_on_information o information_of_vision SubPropertyOf: contributes_to_vision

Figure 46: The Protégé configuration of the super property that the reasoner uses to infer the $contributes\_to\_vision$ object property.

Figure 47 presents the super property that infers the $contributes\_to\_vision$ and $has\_insight$ object properties automatically if we can trace an $Insight$ to a $Vision$ using the chain of object properties.

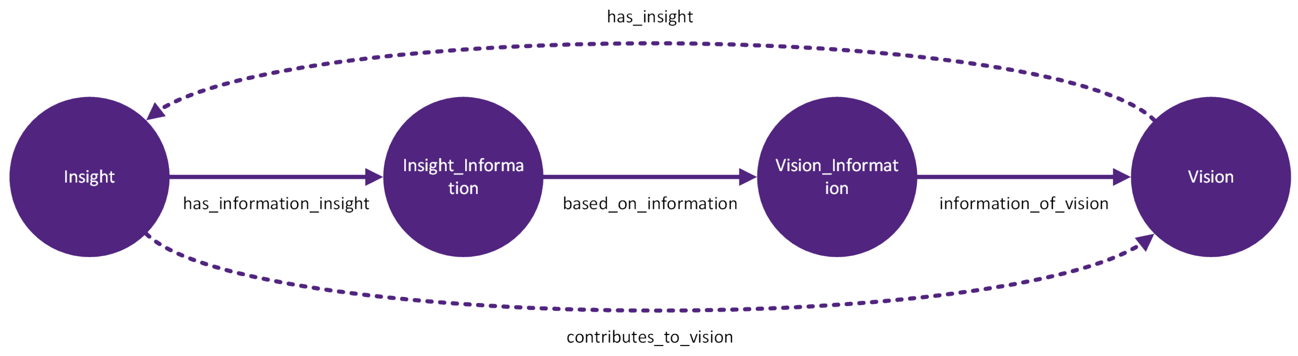


Figure 47: The super property infers the *contributes_to_vision* and *has_insight* object properties automatically if there is a trace from an *Insight* to a *Vision* using the chain of object properties. The reasoner infers the dotted lines using the super property.

Code sample 30 presents the SHACL shapes we use to detect that an *Insight* does not contribute to a *Vision*. Without the super property figure 47 presents the constraints would need to validate three object properties on three target classes. In this case, the super property enables the reasoner to infer these three object properties into a single object property. This example shows that inferencing reduces the complexity of the constraints.

```
rp:Addressed_InsightShape a sh:NodeShape;
   sh:targetClass rp:Addressed_Insight;
   sh:property [
      sh:path rp:contributes_to_vision;
      sh:severity sh:Violation;
      sh:minCount 1;
      sh:message "Structural: Insight does not contribute to a Vision." ; ];
```

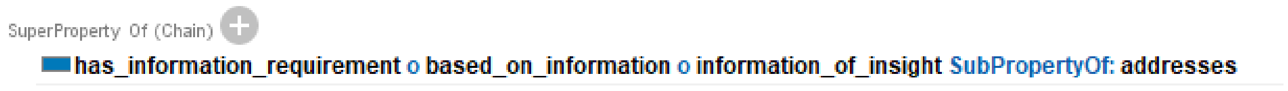
Code sample 30: The SHACL shapes we use to detect that an *Addressed_Insight* does not contribute to a *Vision*.

Structural validation: *Requirement* addresses *Insight*

We want the software product manager to focus on the relevant individuals. Therefore, the constraints should only generate violations for individuals that are relevant for a decision. For example, we need to prevent that an *Insight* that is not addressed by a *Requirement* generates a violation. This *Insight* is not relevant yet. However, the constraints should generate a violation as soon as a *Requirement* addresses this *Insight*.

First, we use the *Addressed_Insight* sub-class to identify the insights that a requirement addresses. Insights that a requirement does not address are not relevant for requirements prioritisation. We use the same concept for the *Addressed_Opportunity* and the *Addressed_Challenge*. The *Addressed_Opportunity* and the *Addressed_Challenge* are subclasses of the *Addressed_Insight* as well.

We also want to make sure every requirement that a software product manager prioritises addresses an insight. The reasoner infers the *addresses* object property from the super property figure 48 presents.



has_information_requirement o based_on_information o information_of_insight SubPropertyOf: addresses

Figure 48: The Protégé configuration of the super property that the reasoner uses to infer the *contributes_to_vision* object property.

Figure 49 presents the super property that infers the *addresses* object property automatically if we can trace a *Requirement* to an *Insight* using the chain of object properties.

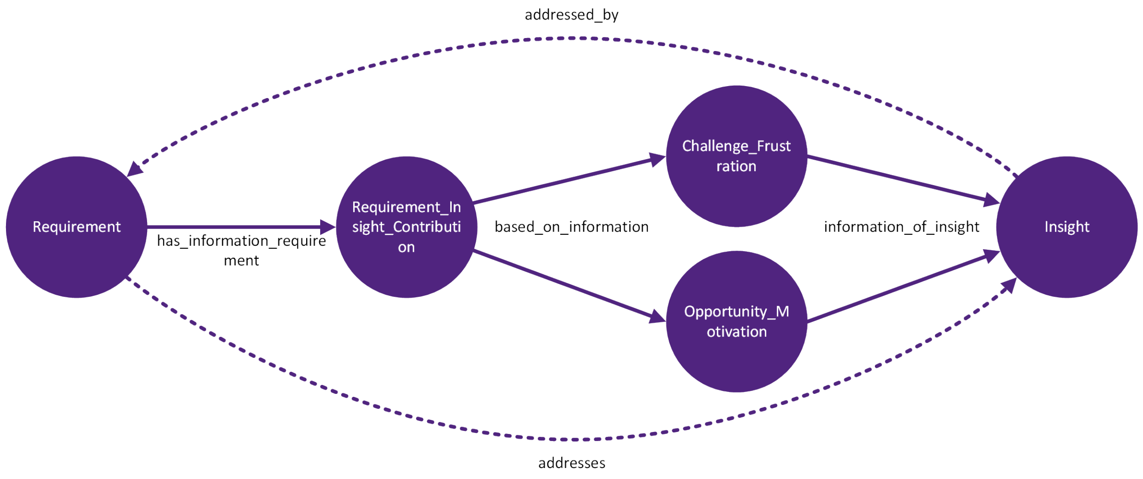


Figure 49: The super property that infers the *addresses* object property automatically if we can trace a *Requirement* to an *Insight* using the chain of object properties.

Code sample 31 presents the SHACL shapes we use to detect that a *Requirement* does not address an *Insight*.

```
rp:RequirementShape a sh:NodeShape;
   sh:targetClass rp:Requirement;
   sh:property [
      sh:path rp:addresses;
      sh:severity sh:Violation;
      sh:minCount 1;
      sh:message "Structural: Requirement does not address an Insight.";
```

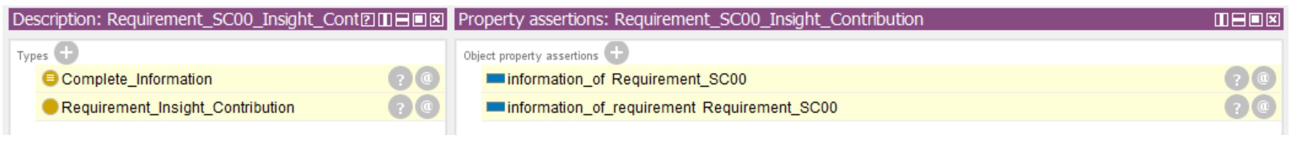
Code sample 31: The SHACL shapes we use to detect that a *Requirement* does not address an *Insight*.

5.1.4 Test scenarios for the decision ontology pattern

We define six abstract test scenarios. The abstraction makes it easier to validate that a specific scenario triggers the expected violations. Scenario $SC0$ validates the structure of the ontology. The first scenario ($SC1$) should not generate any violations. The other scenarios validate a specific pattern and are structurally valid. This structure limits the complexity of the scenarios and their outcomes.

Scenario 0 ($SC0$): Structural validation

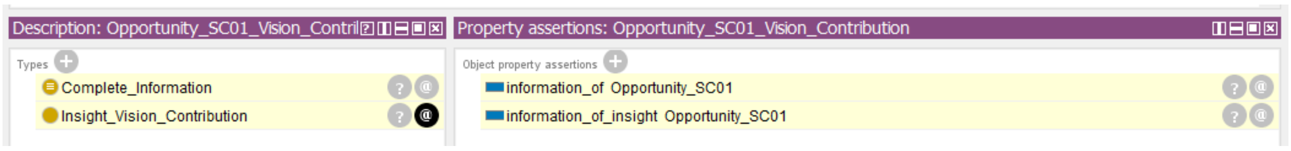
The requirements, insights, and visions we use in the other scenarios are structurally valid. Each *Requirement* addresses at least one *Insight*, and each *Insight* contributes to at least one *Vision*. We create *Requirement_SC0* for this. *Requirement_SC00* does not address an *Insight* and should generate one structural violation and one reproducibility violation. *Requirement_SC00_Insight_Contribution* generates the reproducibility violation as we did not base it on the motivation of frustration of an insight. This missing link breaks the chain figure 49 presents and generates the structural violation. Figure 50 presents the configuration of *Requirement_SC00_Insight_Contribution* in Protégé.


Description: Requirement_SC00_Insight_Cont
Types
Complete_Information
Requirement_Insight_Contribution

Property assertions: Requirement_SC00_Insight_Contribution
Object property assertions
information_of Requirement_SC00
information_of_requirement Requirement_SC00

Figure 50: The individual *Requirement_SC00_Insight_Contribution* in Protégé. The *information_of* object property is the inverse of the *has_information* object property. The reasoner uses this characteristic to infer the *information_of* object property.

Additionally, we define $Requirement\_SC01$. $Requirement\_SC01$ addresses $Opportunity\_SC01$. Therefore $Requirement\_SC01$ should not trigger a structural violation. However, $Opportunity\_SC01$ does not contribute to a $Vision$. $Opportunity\_SC01$ should generate one structural validation and one reproducibility violation. $Opportunity\_SC01\_Vision\_Contribution$ generates the reproducibility violations as we did not base it on vision relevant information. This missing link breaks the chain figure 47 presents and generates the structural violation. Figure 51 presents the configuration of $Opportunity\_SC01\_Vision\_Contribution$ in Protégé.


Description: Opportunity_SC01_Vision_Contri

Types
- Complete_Information
- Insight_Vision_Contribution

Property assertions: Opportunity_SC01_Vision_Contribution

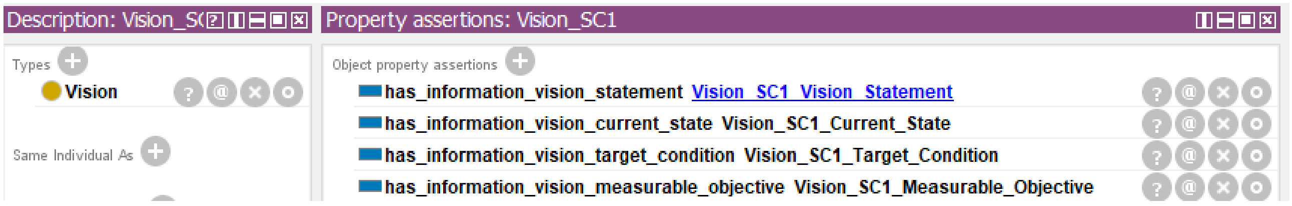
Object property assertions
- information_of Opportunity_SC01
- information_of_insight Opportunity_SC01

Figure 51: The individual $Opportunity\_SC01\_Vision\_Contribution$ in Protégé. The $information\_of$ object property is the inverse of the $has\_information$ object property. The reasoner uses this characteristic to infer the $information\_of$ object property.

We expect 4 structural violations in scenario $SC0$.

Scenario 1 ($SC1$): Decision ontology pattern

The information in the first scenario is complete, reproducible, meets the minimum level of consensus, and does not exceed the maximum level of conflict. Figure 52 presents $Vision\_SC1$ as an example. $Vision\_SC1$ is complete based on the $has\_information\_vision\_current\_state$, $*\_vision\_statement$, $*\_vision\_target\_condition$, and $*\_vision\_measurable\_objective$ object properties.


Description: Vision_S(

Types
- Vision

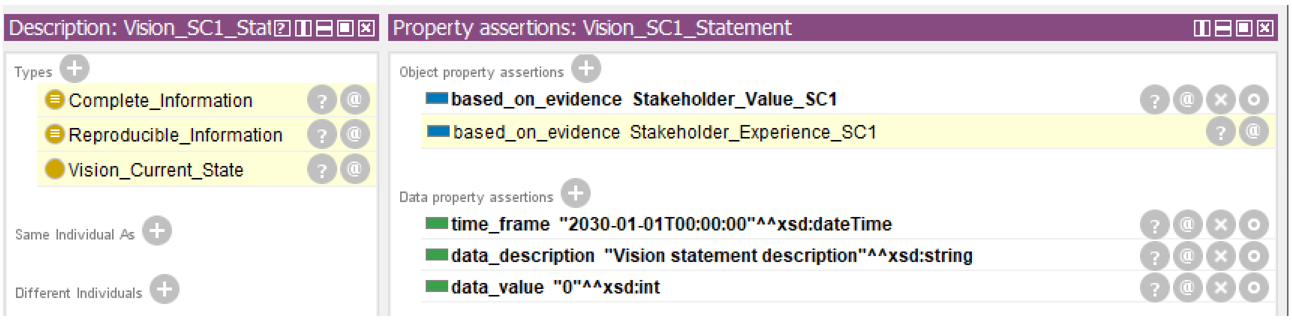
Same Individual As

Property assertions: Vision_SC1

Object property assertions
- has_information_vision_statement Vision_SC1_Vision_Statement
- has_information_vision_current_state Vision_SC1_Current_State
- has_information_vision_target_condition Vision_SC1_Target_Condition
- has_information_vision_measurable_objective Vision_SC1_Measurable_Objective

Figure 52: The individual $Vision\_SC1$ in Protégé. The reasoner infers the $based\_on\_evidence$ object property based on the $agrees\_with$ object property between $Stakeholder\_Value\_SC1$ and $Stakeholder\_Experience\_SC1$.

The completeness pattern requires that each individual classified as $Information$ hosts the $data\_value$ and $data\_description$ data properties. Additionally, the vision statement should also include the $time\_frame$ data property. The reproducibility pattern requires that individuals classified as $Information$ are $based\_on\_evidence$ or $based\_on\_information$. The conflict pattern requires that $Information$ individuals do not have conflicting evidence (represented by the object property $conflict\_with\_evidence$). Figure 53 presents $Vision\_SC1\_Statement$ as an example of a complete and reproducible vision statement. $Vision\_SC1\_Statement$ meets the minimum consensus-level and does not have conflicting evidence.


Description: Vision_SC1_Stat

Types
- Complete_Information
- Reproducible_Information
- Vision_Current_State

Same Individual As

Different Individuals

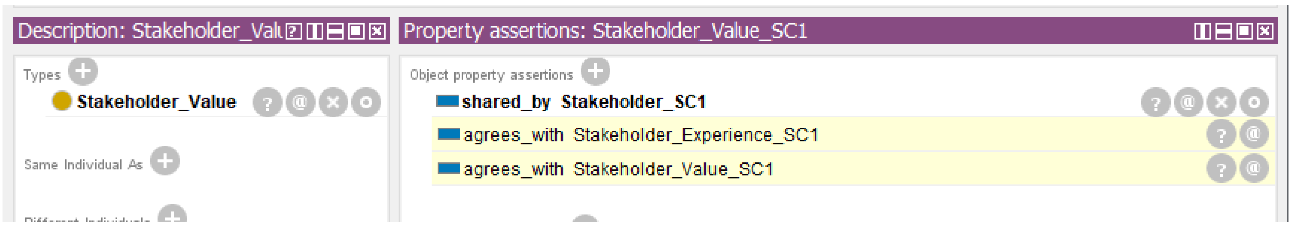
Property assertions: Vision_SC1_Statement

Object property assertions
- based_on_evidence Stakeholder_Value_SC1
- based_on_evidence Stakeholder_Experience_SC1

Data property assertions
- time_frame "2030-01-01T00:00:00"^^xsd:dateTime
- data_description "Vision statement description"^^xsd:string
- data_value "0"^^xsd:int

Figure 53: The individual $Vision\_SC1\_Statement$ in Protégé. The reasoner infers the $based\_on\_evidence$ object properties using the $agrees\_with$ object property between $Stakeholder\_Value\_SC1$ and $Stakeholder\_Experience\_SC1$.

The consensus pattern requires that each individual classified as *Evidence* agrees with at least one other evidence source. Figure 54 presents *Stakeholder_Value_SC1*. *Stakeholder_Value_SC1* agrees with *Stakeholder_Experience_SC1*. The reasoner infers that *Vision_SC1_Statement based_on_evidence Stakeholder_Experience_SC1* from this. Figure 53 presents the inferred object property.


Description: Stakeholder_Valu

Types

Stakeholder_Value

Same Individual As

Property assertions: Stakeholder_Value_SC1

Object property assertions

shared_by Stakeholder_SC1

agrees_with Stakeholder_Experience_SC1

agrees_with Stakeholder_Value_SC1

Figure 54: The individual *Stakeholder_Value_SC1* in Protégé. *Stakeholder_Value_SC1* agrees with *Stakeholder_Experience_SC1*. The reasoner infers that *Vision_SC1_Statement based_on_evidence Stakeholder_Experience_SC1* from this.

We expect 0 violations of the completeness pattern in scenario *SC1*.

Scenario 2 (*SC2*): Completeness pattern

In the second scenario, we attempt to detect incomplete information. We cannot reproduce missing information, and we cannot define a consensus or conflict-level for unreproducible information. As a result, the reproducibility, consensus, and conflict patterns are not relevant in this scenario.

We validate the completeness in two ways. The root classes need to have the specified object properties to the *Information* classes, and the *Information* classes need to have the data properties *data_value* and *data_description*. Figure 55 presents a limited example of the root class *Challenge_SC2* and three of the related *Information* classes. The *Challenge_SC2_Reach* and *Challenge_SC2_Business_Case* are complete. However, the *Insight* itself misses the *Insight_Vision_Contribution* and *Challenge_Frustration*.

Additionally, *Challenge_SC2_Value* misses the *data_value* and *data_description* data properties. We expect six violations related to *Challenge_SC2*. Those six violations cover the missing *Challenge_Frustration*, *Insight_Vision_Contribution*, and their related *data_value* and *data_description*. Additionally, we expect and two violations related to *Challenge_SC2_Value*.

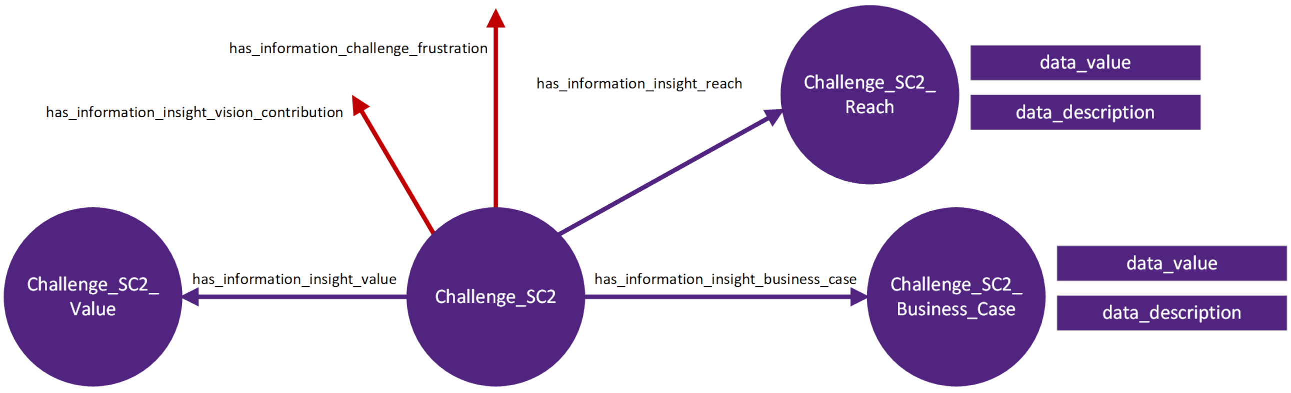
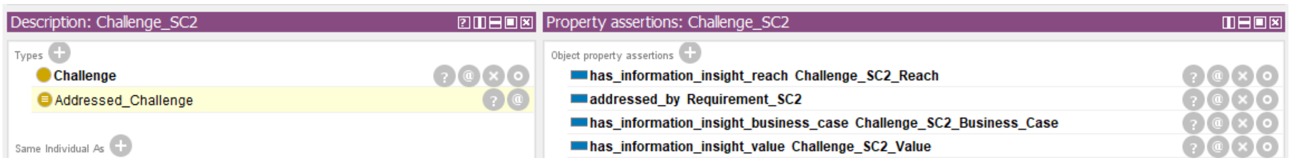

has_information_challenge_frustration
has_information_insight_vision_contribution
has_information_insight_reach
Challenge_SC2_Reach
data_value
data_description
Challenge_SC2_Value
has_information_insight_value
Challenge_SC2
has_information_insight_business_case
Challenge_SC2_Business_Case
data_value
data_description

Figure 55: An incomplete example of the root class *Insight* and three of the related *Information* classes.

Figure 56 presents *Challenge_SC2* in Protégé. The reasoner infers the *Addressed_Challenge* type from the *addressed_by* object property and infers the *has_information* object properties from the specific *has_information_* object properties. For example, the reasoner infers *has_information Challenge_SC2_Value* from *has_information_insight_value Challenge_SC2_Value*.



Description: Challenge_SC2

Types
- Challenge
- Addressed_Challenge

Same Individual As

Property assertions: Challenge_SC2

Object property assertions
- has_information_insight_reach Challenge_SC2_Reach
- addressed_by Requirement_SC2
- has_information_insight_business_case Challenge_SC2_Business_Case
- has_information_insight_value Challenge_SC2_Value

Figure 56: The individual $Challenge\_SC2$ in Protégé. The reasoner infers the $Addressed\_Challenge$ type from the $addressed\_by$ object property and infers the $has\_information$ object properties from the specific $has\_information\_$ object properties.

We defined $Opportunity\_SC2$, $Requirement\_SC2$, and $Vision\_SC2$ as partially incomplete individuals.

$Opportunity\_SC2$ refers to 2 out of the 5 individuals that store information. The 3 missing individuals should generate 3 violations each. Out of these 3 individuals, 1 is complete considering the $data\_value$ and $data\_description$. We expect $Opportunity\_SC2$ to trigger $(3 * 3) + (1 * 2) = 11$ violations.

$Requirement\_SC2$ refers to 4 out of the 10 individuals that store information. The 6 missing individuals should generate 3 violations each. Out of these 3 individuals, 2 are complete considering the $data\_value$ and $data\_description$. We expect $Requirement\_SC2$ to trigger $(6 * 3) + (2 * 2) = 22$ violations.

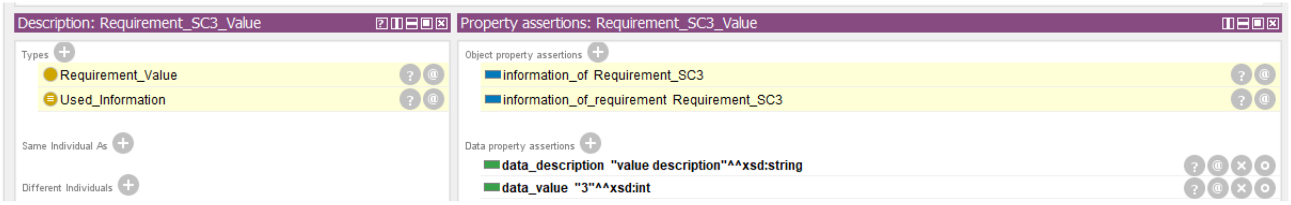
$Vision\_SC2$ refers to 1 out of the 4 individuals that store information. The measurable objective is among the missing 3 individuals. The measurable objective should generate 4 violations: 1 for the individual itself, 1 for the missing $data\_value$, 1 for the missing $data\_description$, and 1 for the missing $time\_frame$. The other missing individuals should generate 3 violations each. The individuals are complete considering the $data\_value$ and $data\_description$. We expect $Vision\_SC2$ to trigger $4 + (2 * 3) + (0 * 2) = 10$ violations.

We expect 43 violations of the completeness pattern in scenario $SC2$.

Scenario 3 ($SC3$): Reproducibility

Scenario $SC3$ contains information that is complete and unreproducible. We cannot define the consensus and conflict-level when information is not reproducible. As a result, the consensus and conflict patterns are not relevant in this scenario.

We create 23 individuals that are not reproducible. These individuals are not evidence-based and are not related to $Information$ using the $based\_on\_information$ object property. For example, $Requirement\_SC3\_Value$ contains the $data\_value$ and $data\_description$ data properties. Therefore, $Requirement\_SC3\_Value$ is complete. However, $Requirement\_SC3\_Value$ is not related to evidence or information. We expect 1 violation related to $Requirement\_SC3\_Value$. Figure 57 presents $Requirement\_SC3\_Value$ in Protégé.



Description: Requirement_SC3_Value

Types
- Requirement_Value
- Used_Information

Same Individual As

Different Individuals

Property assertions: Requirement_SC3_Value

Object property assertions
- information_of Requirement_SC3
- information_of_requirement Requirement_SC3

Data property assertions
- data_description "value description"^^xsd:string
- data_value "3"^^xsd:int

Figure 57: The individual $Requirement\_SC3\_Value$ in Protégé. The $information\_of$ object property is the inverse of the $based\_on\_information$ object property. $Requirement\_SC3$ is based on $Requirement\_SC3\_Value$. The reasoner infers the $information\_of$ from its inverse.

We define 4 root individuals for which we validate the reproducibility: $Opportunity\_SC3$, $Challenge\_SC3$, $Requirement\_SC3$, and $Vision\_SC3$. $Opportunity\_SC3$ requires 3 knowledge criteria, $Challenge\_SC3$ requires 3 knowledge criteria as well, and $Requirement\_SC3$ requires 4 knowledge criteria. We base these 10 knowledge criteria on another 13 information sources spread out over $Opportunity\_SC3$, $Challenge\_SC3$, $Requirement\_SC3$, and $Vision\_SC3$. As a result, we expect 23 violations of the reproducibility pattern in scenario $SC3$.

Scenario 4 (*SC4*): Conflict pattern

In scenario *SC4*, we validate that the information is complete, reproducible, and does not cause any conflict. Figure 58 presents how this evidence disagrees with each other. Figure 59 presents *Stakeholder_Experience_SC4* in Protégé.

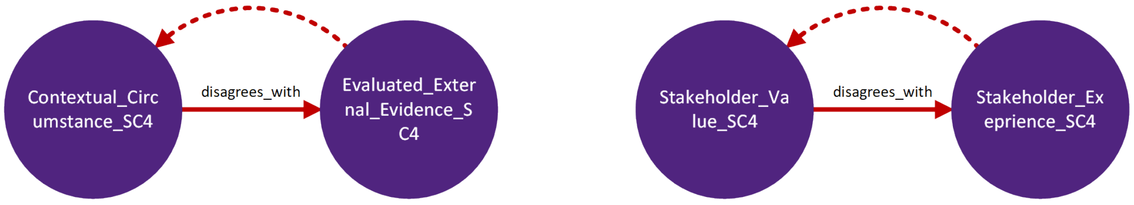
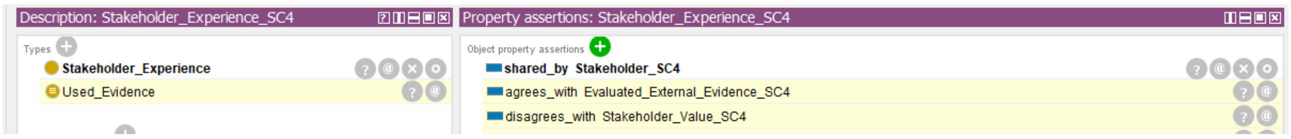

Contextual_Circumstance_SC4
disagrees_with
Evaluated_External_Evidence_SC4
Stakeholder_Value_SC4
disagrees_with
Stakeholder_Experience_SC4

Figure 58: 4 evidence sources disagree with each other. The fixed lines represent configured object properties. The dotted lines represent inferred object properties.


Description: Stakeholder_Experience_SC4

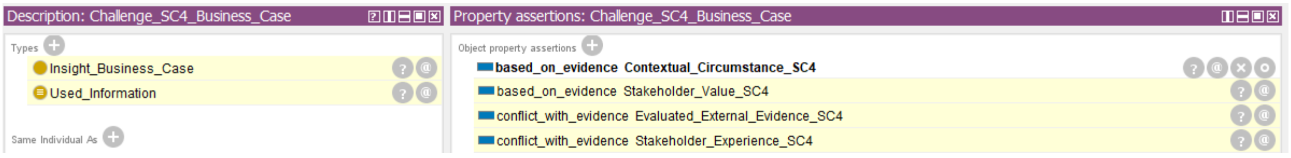
Types
- Stakeholder_Experience
- Used_Evidence

Property assertions: Stakeholder_Experience_SC4

Object property assertions
- shared_by Stakeholder_SC4
- agrees_with Evaluated_External_Evidence_SC4
- disagrees_with Stakeholder_Value_SC4

Figure 59: The individual *Stakeholder_Experience_SC4* in Protégé. The reasoner infers the *disagrees_with* object property using its symmetric and irreflexive characteristics.

For example, we use *Contextual_Circumstance_SC4* as evidence for *Challenge_SC4_Business_Case*. *Contextual_Circumstance_SC4* disagrees with *Evaluated_External_Evidence_SC4*. As a result, *Challenge_SC4_Business_Case* conflicts with these evidence sources. We expect 2 violations related to *Challenge_SC4_Business_Case*: 1 for each evidence conflict. Figure 60 presents *Challenge_SC4_Business_Case* in Protégé.


Description: Challenge_SC4_Business_Case

Types
- Insight_Business_Case
- Used_Information

Same Individual As

Property assertions: Challenge_SC4_Business_Case

Object property assertions
- based_on_evidence Contextual_Circumstance_SC4
- based_on_evidence Stakeholder_Value_SC4
- conflict_with_evidence Evaluated_External_Evidence_SC4
- conflict_with_evidence Stakeholder_Experience_SC4

Figure 60: The individual *Challenge_SC4_Business$_{Case}$* in Protégé. The reasoner infers the *conflict_with_evidence* object property from the super property figure 27 in section 4.3.5 Conflict presents.

We define 4 conflicting evidence sources: *Contextual_Circumstance_SC4*, *Evaluated_External_Evidence_SC4* and *Stakeholder_Value_SC4*, *Stakeholder_Experience_SC4*[5]. We based 16 individuals on these evidence sources: 3 individuals that host information for *Challenge_SC4*, 3 individuals that host information for *Opportunity_SC4*, 6 individuals that host information for, *Requirement_SC4*, and another 4 individuals that host information for *Vision_SC4*. As a result, we expect 16 violations of the conflict pattern in scenario *SC4*.

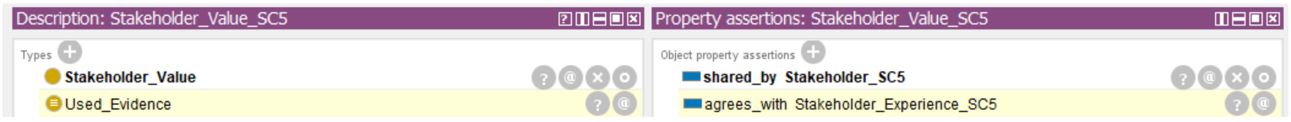
Scenario 5 (*SC5*): Consensus pattern

The last scenario contains information that is complete and reproducible. It does not contain evidence conflicts. However, some information does not meet the defined consensus constraints. Each individual that the reasoner classifies as *Used_Evidence* should have at least one *agrees_with* object property to meet the consensus constraints.

We have created 4 evidence sources for scenario *SC5*: *Contextual_Circumstance_SC5*, *Evaluated_External_Evidence_SC5*, *Stakeholder_Experience_SC5*, and *Stakeholder_Value_SC5*. These

[5]*Stakeholder_Experience_SC4* and *Stakeholder_Value_SC4* are shared by *Stakeholder_SC4* to ensure this evidence does not trigger any reproducibility violations.

evidence sources serve as evidence for the *Information* related to *SC*5, for example, *Opportunity_SC5_Reach*. *Stakeholder_Experience_SC*5 and *Stakeholder_Value_SC*5 agree with each other and should, therefore, not trigger any violations. Figure 61 presents *Stakeholder_Value_SC*5 in Protégé.


Description: Stakeholder_Value_SC5

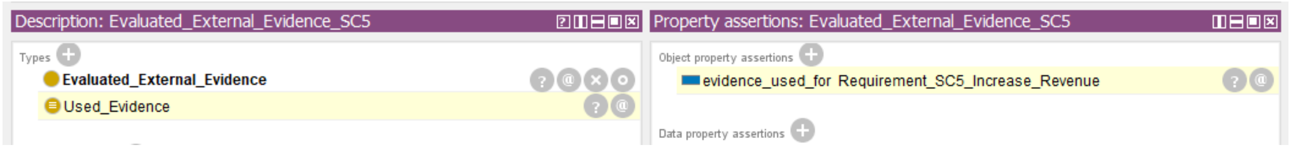
Types
- Stakeholder_Value
- Used_Evidence

Property assertions: Stakeholder_Value_SC5

Object property assertions
- shared_by Stakeholder_SC5
- agrees_with Stakeholder_Experience_SC5

Figure 61: The *Stakeholder_Value_SC*5 in Protégé. The reasoner infers the *agrees_with* object property based its symmetric characteristic.

Figure 62 presents, for example, *Evaluated_External_Evidence_SC*5 in Protégé. *Evaluated_External_Evidence_SC*5 is used as evidence for *Requirement_SC5_Increase_Revenue* and is, therefore, classified as *Used_Evidence*. However, it does not have an *agrees_with* object property and should generate 1 violation. *Contextual_Circumstance_SC*5 does not have an *agrees_with* and should generate 1 violation as well.


Description: Evaluated_External_Evidence_SC5

Types
- Evaluated_External_Evidence
- Used_Evidence

Property assertions: Evaluated_External_Evidence_SC5

Object property assertions
- evidence_used_for Requirement_SC5_Increase_Revenue

Data property assertions

Figure 62: The *Evaluated_External_Evidence_SC*5 in Protégé. The *evidence_used_for* object property is the inverse of the *based_on_evidence* object property. The reasoner uses this inverse characteristic to infer the *evidence_used_for* object property.

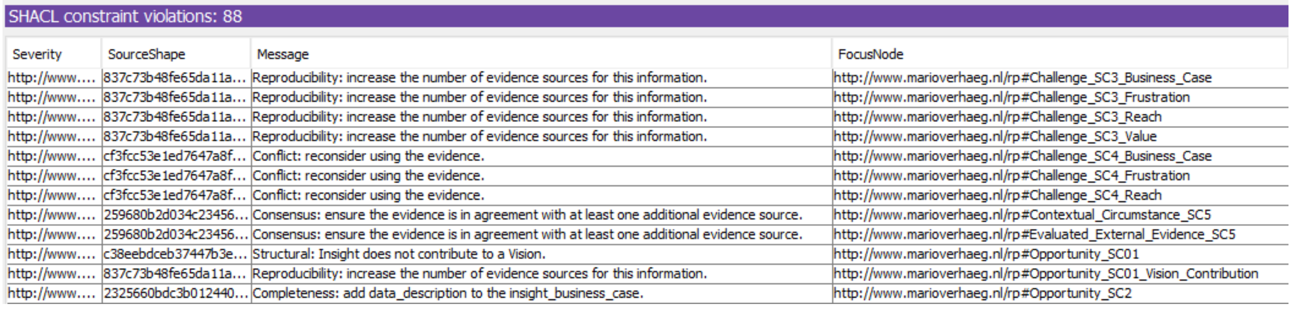
As a result, we expected 2 violations of the consensus pattern in scenario *SC*5.

The result of the test scenarios

Table 12 presents the results of the tests. We detect 88 violations. Figure 63 presents an extract of the results as the SHACL4P plugin presents them in Protégé.

Table 12: The number of expected and detected violations per scenario and pattern.

Scenario	Scenario	Expected violations	Detected violations
*SC*0	Structural	4	4
*SC*1	n/a	0	0
*SC*2	Completeness	43	43
*SC*3	Reproducibility	23	23
*SC*4	Conflict	16	16
*SC*5	Consensus	2	2



SHACL constraint violations: 88

Severity	SourceShape	Message	FocusNode
http://www.…	837c73b48fe65da11a…	Reproducibility: increase the number of evidence sources for this information.	http://www.marioverhaeg.nl/rp#Challenge_SC3_Business_Case
http://www.…	837c73b48fe65da11a…	Reproducibility: increase the number of evidence sources for this information.	http://www.marioverhaeg.nl/rp#Challenge_SC3_Frustration
http://www.…	837c73b48fe65da11a…	Reproducibility: increase the number of evidence sources for this information.	http://www.marioverhaeg.nl/rp#Challenge_SC3_Reach
http://www.…	837c73b48fe65da11a…	Reproducibility: increase the number of evidence sources for this information.	http://www.marioverhaeg.nl/rp#Challenge_SC3_Value
http://www.…	cf3fcc53e1ed7647a8f…	Conflict: reconsider using the evidence.	http://www.marioverhaeg.nl/rp#Challenge_SC4_Business_Case
http://www.…	cf3fcc53e1ed7647a8f…	Conflict: reconsider using the evidence.	http://www.marioverhaeg.nl/rp#Challenge_SC4_Frustration
http://www.…	cf3fcc53e1ed7647a8f…	Conflict: reconsider using the evidence.	http://www.marioverhaeg.nl/rp#Challenge_SC4_Reach
http://www.…	259680b2d034c23456…	Consensus: ensure the evidence is in agreement with at least one additional evidence source.	http://www.marioverhaeg.nl/rp#Contextual_Circumstance_SC5
http://www.…	259680b2d034c23456…	Consensus: ensure the evidence is in agreement with at least one additional evidence source.	http://www.marioverhaeg.nl/rp#Evaluated_External_Evidence_SC5
http://www.…	c38eebdceb37447b3e…	Structural: Insight does not contribute to a Vision.	http://www.marioverhaeg.nl/rp#Opportunity_SC01
http://www.…	837c73b48fe65da11a…	Reproducibility: increase the number of evidence sources for this information.	http://www.marioverhaeg.nl/rp#Opportunity_SC01_Vision_Contribution
http://www.…	2325660bdc3b012440…	Completeness: add data_description to the insight_business_case.	http://www.marioverhaeg.nl/rp#Opportunity_SC2

Figure 63: An extract of the results as the SHACL4P plugin presents them in Protégé.

5.1.5 Decision-relevant root individuals

We determine the decision root individual that are relevant for a specific requirements prioritisation decision. The requirement prioritisation decision requires two individuals that are both classified as *Requirement*: r_1 and r_2. We know that a requirement needs to address an insight and that an insight has two sub-classes: opportunity and challenge. We also know that a vision should reproduce an insight. Code sample 32 presents a SPARQL query that retrieves all opportunities, challenges, and visions related to a requirement using parameters $< r1 >$ and $< r2 >$. We execute the SPARQL query and feed the results into the set RI. We define function $rpri(r_1, r_2) = RI$ to logically represent the SPARQL query. We manually add r_1 and r_2 to RI.

```
SELECT DISTINCT ?ri
WHERE
{
   # Gather insights (opportunities and challenges)
   {
      ?req rp:addresses ?ri .
      ?ri rdf:type ?t
      FILTER(?t != owl:Thing && ?t != rp:Addressed_Insight &&  ?t != rp:Addressed_Opportunity && ?t != rp:Addressed_Challenge && ?t != rp:Insight).
   }
   UNION
   # Gather visions
   {
      ?req rp:addresses ?ins .
      ?ins rp:has_information ?ivc .
      ?ivc rdf:type rp:Insight_Vision_Contribution .
      ?ivc rp:based_on_information ?vcs .
      ?ri rp:has_information ?vcs .
      ?ri rdf:type ?t .
      FILTER(?t != owl:Thing).
   }
   FILTER(?req = <r1> || ?req = <r2>)
}
```

Code sample 32: The first part of the SPARQL query gathers the insight(s) related to the specified requirements based on the *addresses* object property: a *Requirement addresses* an insight. The second part of the SPARQL query gathers the vision(s) related to the specified requirements based on the *addresses*, *has_information*, and *based_on_information* object properties.

5.1.6 Test scenarios for the decision presentation pattern

We use four requirements to validate the decision presentation pattern. Each requirement has a different information maturity-level. The software product manager prioritises *Requirement_SC1* and *Requirement_SC2* in test scenario *DEC1*. The software product manager prioritises *Requirement_SC3* and *Requirement_SC4* in test scenario *DEC2*.

Decision $DEC1$**:** $Requirement\_SC1$ **versus** $Requirement\_SC2$

A software product manager needs to decide if $Requirement\_SC1$ is more important than $Requirement\_SC2$. The first question we ask is:

Is the information ready for an evidence-based decision?

The first dashboard of the decision presentation pattern helps the decision-maker to answer this question. We define the consolidated information maturity-level, and the evidence spread for these two requirements to generate this dashboard.

We use function $rpri(Requirement\_SC1, Requirement\_SC2)$ to define RI_{DEC1}. RI_{DEC1} is the set of decision-relevant root individuals. RI_{DEC1} includes $Challenge\_SC1$, $Opportunity\_SC1$, $Challenge\_SC2$, $Opportunity\_SC2$, $Vision\_SC1$, and $Vision\_SC2$. We manually add $Requirement\_SC1$ and $Requirement\_SC2$ to RI_{DEC1}. Table 13 presents the maximum and the actual number of violations per decision-relevant root individual. We conclude that decision $DEC1$ can generate up to 239 violations, and that decision $DEC1$ generates 52 violations.

Table 13: The maximum and the actual number of violations per decision-relevant root individual.

Function	mvi_1	mvi_2	mvi_3	mvi_4	$Total_{mv}$	avi_1	avi_2	avi_3	avi_4	$Total_{av}$
$Requirement\_SC1$	30	10	2	10	52	0	0	0	0	0
$Challenge\_SC1$	15	5	2	5	27	0	0	0	0	0
$Opportunity\_SC1$	15	5	2	5	27	0	0	0	0	0
$Vision\_SC1$	14	4	2	4	24	0	0	0	0	0
$Requirement\_SC2$	30	10	4	10	54	23	0	0	0	22
$Opportunity\_SC2$	15	5	4	5	29	11	0	0	0	11
$Vision\_SC2$	14	4	4	4	26	10	0	0	0	10
Total	133	43	20	43	239	44	0	0	0	44

The information table 13 presents allows us to calculate the information maturity-level iml for decision $DEC1$. Equation 12 shows that the information maturity-level for $DEC1$ is 82%.

$$iml(RI) = \frac{mv_(RI) - av(RI)}{mv(RI)} = \frac{239 - 44}{239} = 0.816 \qquad (12)$$

The first dashboard also presents the evidence spread. Table 14 presents the evidence spread for decision $DEC1$ using the SPARQL query code sample 17 presents.

Table 14: The evidence spread for the decision-relevant root individuals and the total evidence spread for this scenario.

Evidence	$Requirement\_SC1$	$Requirement\_SC2$	Total
$Contextual\_Circumstance$	9	3	12
$Stakeholder\_Value$	7	3	10
$Evaluated\_External\_Evidence$	9	3	12
$Stakeholder\_Experience$	7	3	10

Figure 64 presents a mock-up of the first decision presentation pattern dashboard. The dashboard reflects an information maturity-level of 82% and the evidence spread table 14 presents. The decision-maker needs to decide if the information maturity-level and evidence-spread are acceptable, depending on the impact of the decision.

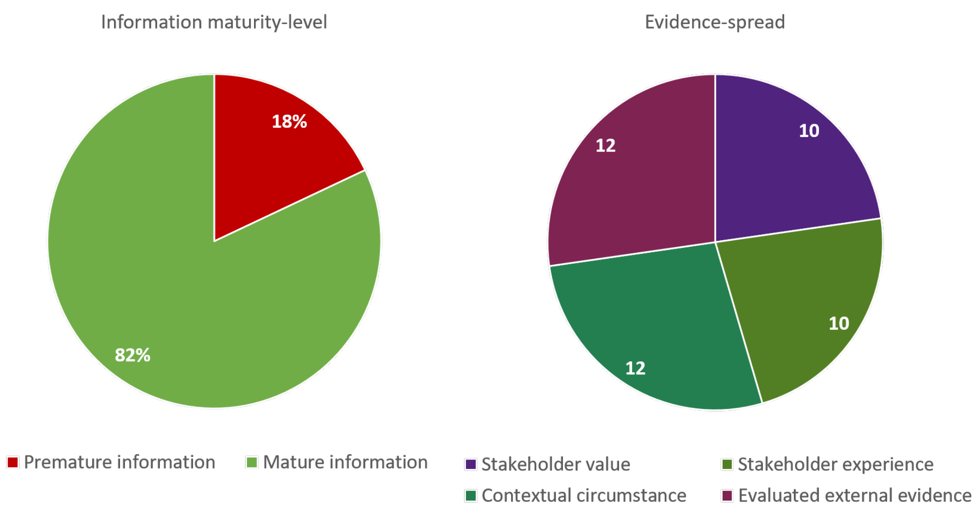


Figure 64: A mock-up of the first decision presentation pattern dashboard for decision $DEC1$. The dashboard reflects an information maturity-level of 82% and the evidence spread table 14 presents.

The second dashboard of the decision presentation pattern presents the information maturity-level per pattern. We present the pattern-specific maximum and the actual violations in table 13.

$Requirement\_SC1$ does not generate any completeness violations. $Requirement\_SC2$ generates 44 completeness violations. The completeness pattern can generate up to 133 violations in total for decision $DEC1$. $Requirement\_SC1$ and $Requirement\_SC2$ do not generate any other violations.

Figure 65 presents a mock-up of the second decision presentation pattern dashboard. The dashboard reflects the completeness maturity-level of $\frac{133-44}{133} = 0.669$, which results in 67%. The maturity-levels of the other patterns are 100% as they do not generate any violations.

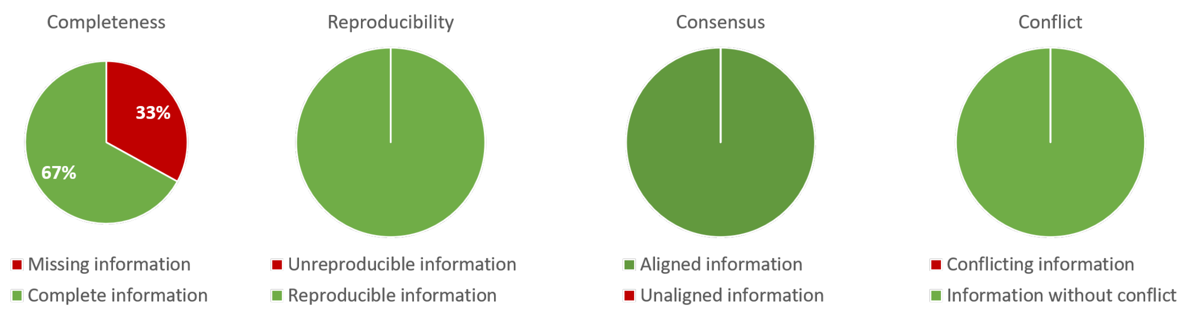
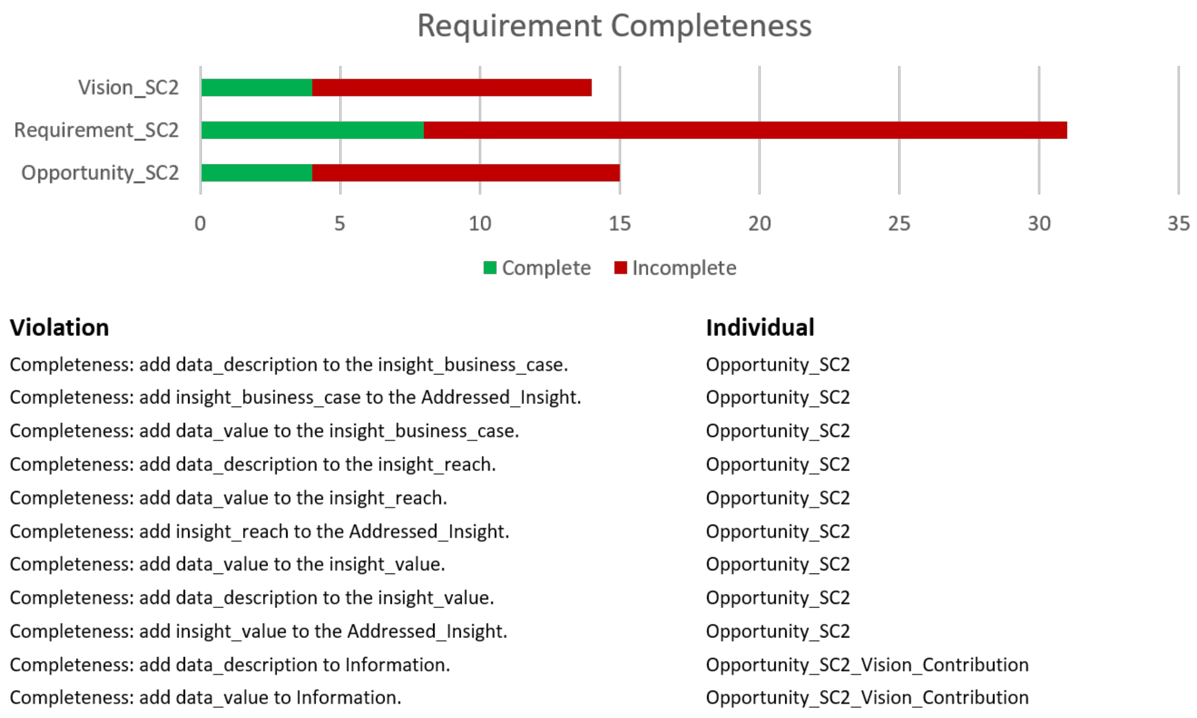


Figure 65: A mock-up of the second decision presentation pattern dashboard for decision $DEC1$. The dashboard reflects a completeness maturity-level of 67%.

To complete the missing information, the software product manager needs to know which information is incomplete. The third dashboard of the decision presentation pattern presents the completeness of information per decision-relevant individual.

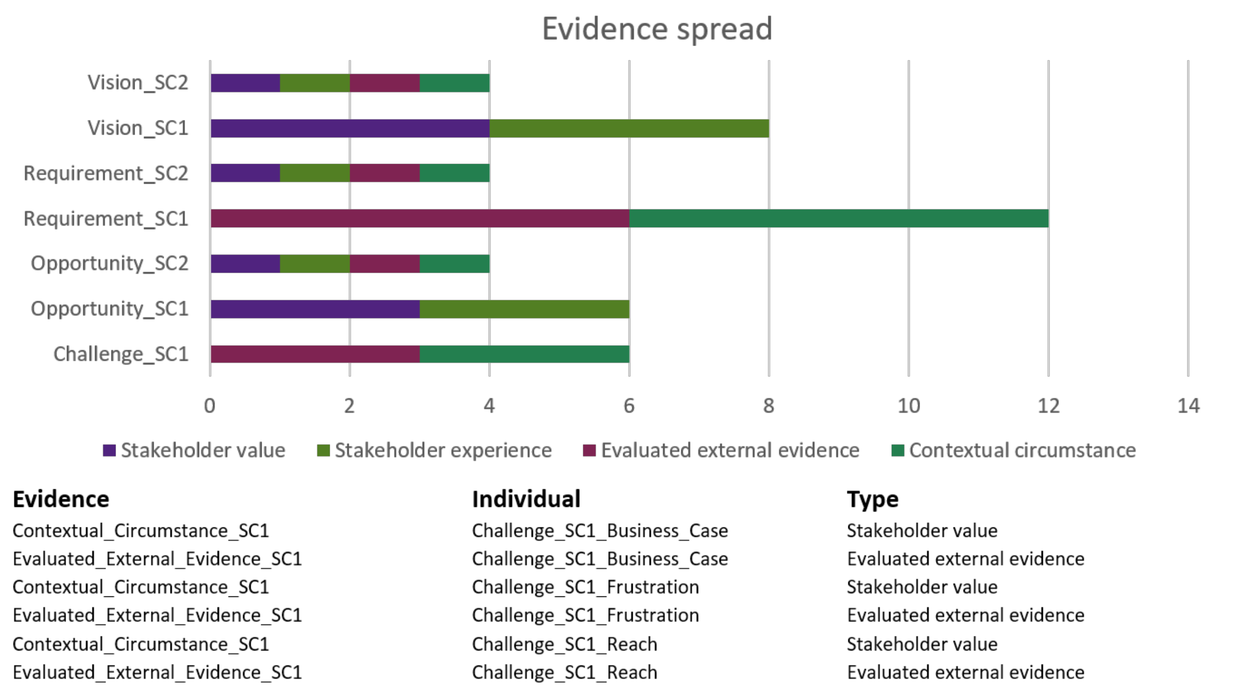
Figure 66 presents a mock-up of the third decision presentation pattern dashboard, considering the software product manager selects $Opportunity\_SC2$. We observe that three decision-relevant individuals are incomplete. The bar charts give the software product manager an indication which individuals are causing the 67% completeness maturity-level. The software product manager wants to know more information on a specific individual: $Opportunity\_SC2$. The dashboard presents the violations related to $Opportunity\_SC2$ in the table below the bar chart. We also observe the consolidation into root individuals: individuals classified as $Vision$, $Requirement$, $Opportunity$, or $Challenge$ are root individuals. The dashboard consolidates the violations of $Opportunity\_SC2\_Value$ under $Opportunity\_SC2$.



Violation	**Individual**
Completeness: add data_description to the insight_business_case.	Opportunity_SC2
Completeness: add insight_business_case to the Addressed_Insight.	Opportunity_SC2
Completeness: add data_value to the insight_business_case.	Opportunity_SC2
Completeness: add data_description to the insight_reach.	Opportunity_SC2
Completeness: add data_value to the insight_reach.	Opportunity_SC2
Completeness: add insight_reach to the Addressed_Insight.	Opportunity_SC2
Completeness: add data_value to the insight_value.	Opportunity_SC2
Completeness: add data_description to the insight_value.	Opportunity_SC2
Completeness: add insight_value to the Addressed_Insight.	Opportunity_SC2
Completeness: add data_description to Information.	Opportunity_SC2_Vision_Contribution
Completeness: add data_value to Information.	Opportunity_SC2_Vision_Contribution

Figure 66: A mock-up of the third decision presentation pattern dashboard, considering the decision-maker selected $Opportunity\_SC2$. We observe that four decision-relevant individuals are incomplete.

The software product manager wants to understand the reproducibility of the information. Figure 67 presents the evidence spread using the third dashboard, considering the decision-maker clicked on the pie chart presenting the evidence spread in the first dashboard. This evidence spread does not violate any constraints. However, the software product manager can improve the evidence spread by elaborating on, for example, $Vision\_SC1$, which is based on two out of four evidence types.



Evidence	Individual	Type
Contextual_Circumstance_SC1	Challenge_SC1_Business_Case	Stakeholder value
Evaluated_External_Evidence_SC1	Challenge_SC1_Business_Case	Evaluated external evidence
Contextual_Circumstance_SC1	Challenge_SC1_Frustration	Stakeholder value
Evaluated_External_Evidence_SC1	Challenge_SC1_Frustration	Evaluated external evidence
Contextual_Circumstance_SC1	Challenge_SC1_Reach	Stakeholder value
Evaluated_External_Evidence_SC1	Challenge_SC1_Reach	Evaluated external evidence

Figure 67: The evidence spread using the third dashboard, considering the decision-maker selected $Challenge\_SC1$. We observe that, for example, $Stakeholder\_Value$ and $Stakeholder\_Experience$ reproduce $Vision\_SC1$.

Decision $DEC2$: $Requirement\_SC3$ versus $Requirement\_SC4$

A software product manager needs to decide if $Requirement\_SC3$ is more important than $Requirement\_SC4$. The first question we ask is:

Is the information ready for this decision?

The first decision presentation pattern dashboard helps a decision-maker to answer this question. We define the consolidated information maturity-level, and the evidence spread for the two requirements to generate this dashboard.

We use function $rpri(Requirement\_SC3, Requirement\_SC4)$ to define RI_{DEC2}. RI_{DEC2} is the set of decision-relevant root individuals. RI_{DEC2} includes $Challenge\_SC3$, $Opportunity\_SC3$, $Challenge\_SC4$, $Opportunity\_SC4$, $Vision\_SC3$, and $Vision\_SC4$. We manually add $Requirement\_SC3$ and $Requirement\_SC4$ to RI_{DEC2}. Table 15 presents the maximum and the actual number of violations per decision-relevant root individual. The bottom line of the table presents the maximum and the actual number of violations per pattern. We conclude that decision $DEC2$ can generate up to 252 violations. We conclude that decision $DEC2$ generates 37 violations.

Table 15: The maximum number of violations per decision-relevant root individual.

Function	mvi_1	mvi_2	mvi_3	mvi_4	$Total_{mv}$	avi_1	avi_2	avi_3	avi_4	$Total_{av}$
Requirement_SC3	30	10	0	10	50	0	9	0	0	9
Challenge_SC3	15	5	0	5	25	0	4	0	0	4
Opportunity_SC3	15	5	0	5	25	0	4	0	0	4
Requirement_SC4	30	10	2	10	52	0	0	0	6	6
Challenge_SC4	15	5	2	5	27	0	0	0	3	3
Opportunity_SC4	15	5	2	5	27	0	0	0	3	3
Vision_SC3	14	4	0	4	22	0	4	0	0	4
Vision_SC4	14	4	2	4	24	0	0	0	4	4
Total	148	48	8	48	252	0	21	0	16	37

The information presented in table 15 allows us to calculate the information maturity-level iml for decision $DEC2$. Equation 13 shows that the information maturity-level for $DEC2$ is 85%.

$$iml(RI) = \frac{mv_(RI) - av(RI)}{mv(RI)} = \frac{252 - 37}{252} = 0.853 \tag{13}$$

The first dashboard also presents the evidence spread. Table 16 presents the evidence spread for decision $DEC2$ using the SPARQL query code sample 17 presents.

Table 16: The evidence spread for the decision-relevant root individuals and the total evidence spread for this scenario.

Evidence	*Requirement_SC3*	*Requirement_SC4*	Total
Contextual_Circumstance	0	13	13
Stakeholder_Value	0	13	13
Evaluated_External_Evidence	0	3	3
Stakeholder_Experience	0	3	3

Figure 68 presents a mock-up of the decision presentation pattern first dashboard. The dashboard reflects an information maturity-level of 85% and the evidence spread table 16 presents. The decision-maker needs to decide if the information maturity-level and evidence-spread are acceptable, depending on the impact of the decision.

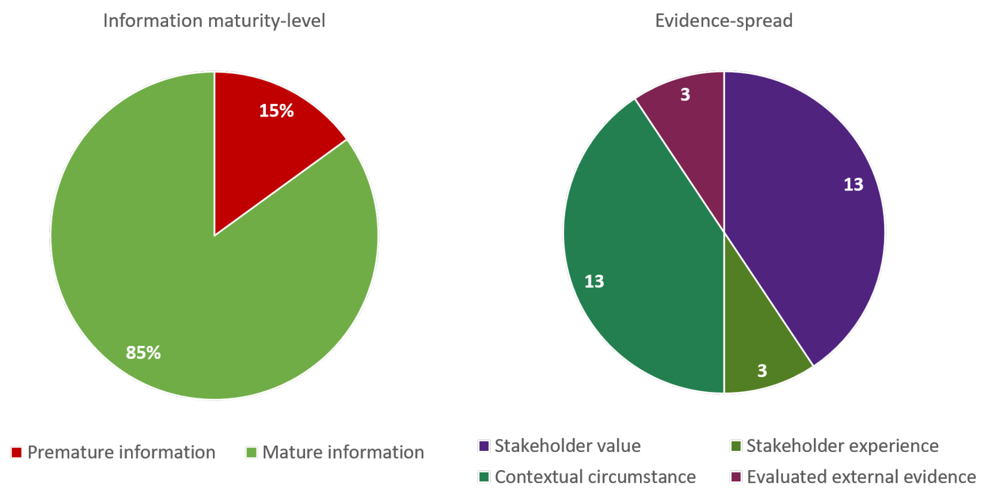


Figure 68: A mock-up of the first decision presentation pattern dashboard for test scenario $DEC2$. The dashboard reflects an information maturity-level of 85% and the evidence spread table 16 presents.

The second decision presentation pattern dashboard presents the information maturity-level per pattern. We present the pattern-specific maximum and the actual violations in table 15.

$Requirement\_SC3$ and $Requirement\_SC4$ do not generate any completeness violations. We use $Requirement\_SC3$ to test the reproducibility pattern, and it generates 21 reproducibility violations. The reproducibility pattern can generate up to 48 violations for decision $DEC2$. We use $Requirement\_SC4$ to test the conflict pattern, and it generates 16 conflict violations. The conflict pattern can generate up to 48 violations for decision $DEC2$. $Requirement\_SC3$ and $Requirement\_SC4$ do not generate any other violations.

Figure 69 presents a mock-up of the second decision presentation pattern dashboard. The dashboard reflects the reproducibility maturity-level of $\frac{48-21}{48} = 0.563$, which results in 56% and the conflict maturity-level of $\frac{48-16}{48} = 0.667$, which results in 67%. The maturity-levels of the other patterns are 100% as they do not generate any violations.

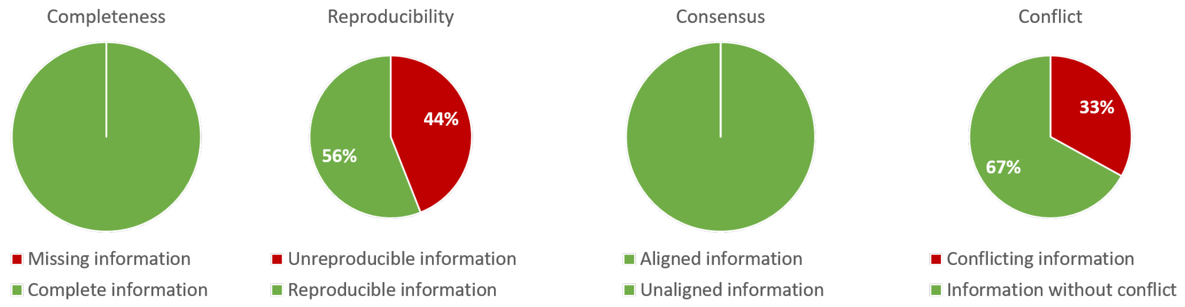
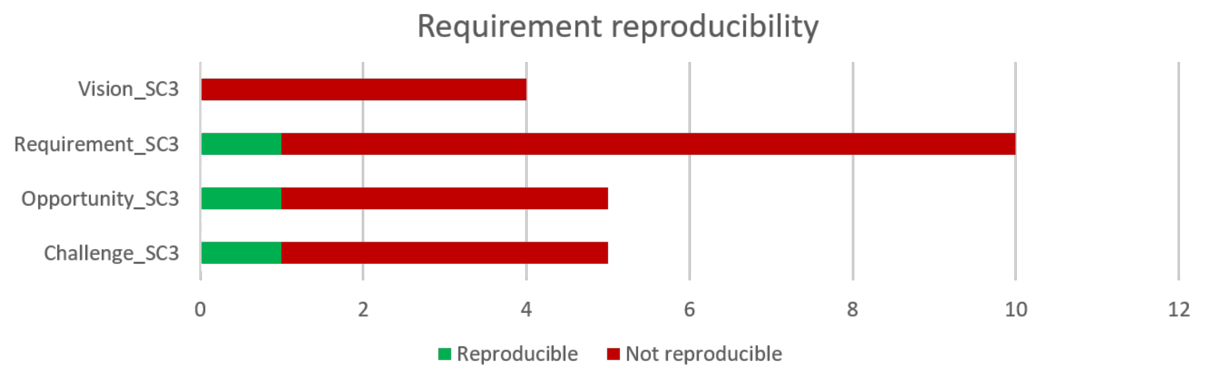


Figure 69: A mock-up of the second decision presentation pattern dashboard for decision $DEC2$. The dashboard reflects a reproducibility maturity-level of 56% and a conflict maturity-level of 67%.

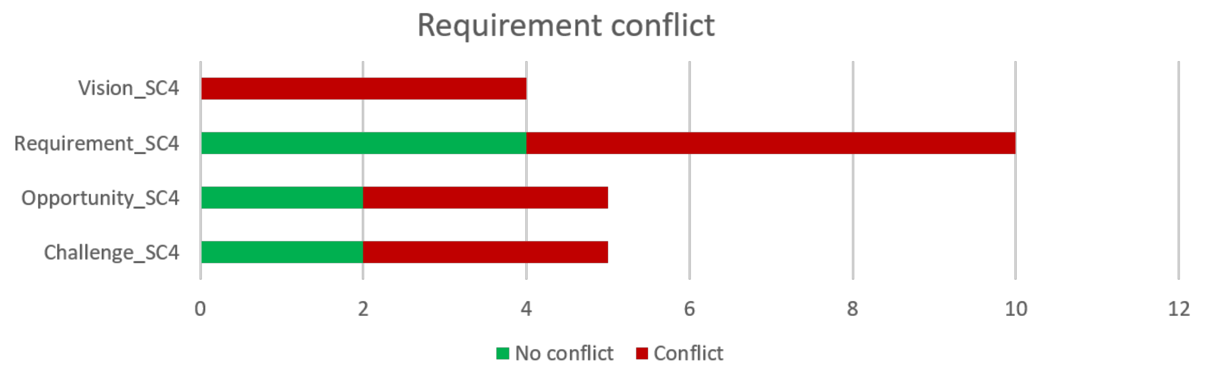
Figure 70 presents a mock-up of the third decision presentation pattern dashboard, considering the software product manager selected $Vision\_SC3$ and clicked on the pie chart presenting the reproducibility maturity-level in the second dashboard. We observe that four decision-relevant individuals are not reproducible. The bar charts give the software product manager an indication which individuals are causing the 57% reproducibility maturity-level. The software product manager wants to know more information on a specific individual: $Vision\_SC3$. The dashboard presents the violations related to $Vision\_SC3$ in the table below the bar chart. We also observe the consolidation into root individuals: individuals classified as $Vision$, $Requirement$, $Opportunity$, or $Challenge$ are root individuals.



Violation	**Individual**
Reproducibility: increase the number of evidence sources for this information.	Vision_SC3_Current_State
Reproducibility: increase the number of evidence sources for this information.	Vision_SC3_Measurable_Objective
Reproducibility: increase the number of evidence sources for this information.	Vision_SC3_Target_Condition
Reproducibility: increase the number of evidence sources for this information.	Vision_SC3_Vision_Statement

Figure 70: A mock-up of the third decision presentation pattern dashboard, considering the decision-maker selected *Vision_SC3*. We observe that four decision-relevant individuals are incomplete.

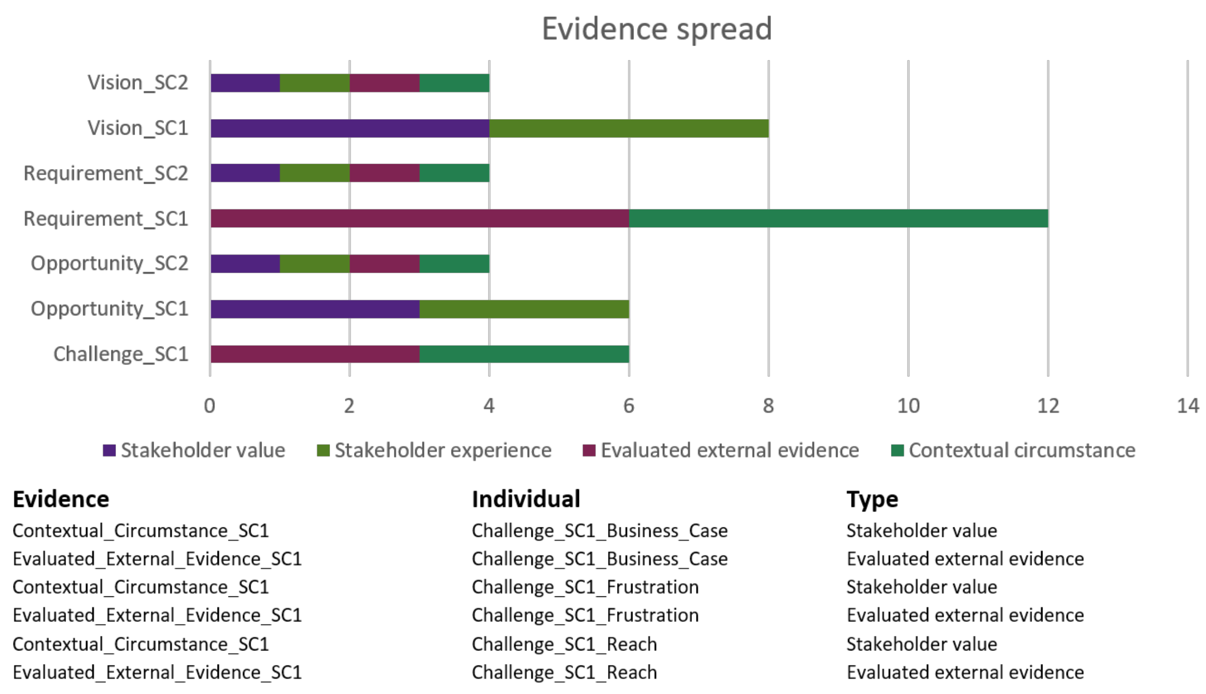
Figure 71 presents a mock-up of the third decision presentation pattern dashboard, considering the software product manager selected *Opportunity_SC4* and clicked on the pie chart presenting the conflict maturity-level in the second dashboard.



Violation	**Individual**
Conflict: reconsider using the evidence.	Opportunity_SC4_Business_Case
Conflict: reconsider using the evidence.	Opportunity_SC4_Motivation
Conflict: reconsider using the evidence.	Opportunity_SC4_Reach

Figure 71: A mock-up of the third decision presentation pattern dashboard, considering the decision-maker selected *Opportunity_SC4*.

The software product manager wants to understand the reproducibility of the information. Figure 72 presents the evidence spread using the third dashboard, considering the decision-maker clicked on the pie chart presenting the evidence spread in the first dashboard.



Evidence	Individual	Type
Contextual_Circumstance_SC1	Challenge_SC1_Business_Case	Stakeholder value
Evaluated_External_Evidence_SC1	Challenge_SC1_Business_Case	Evaluated external evidence
Contextual_Circumstance_SC1	Challenge_SC1_Frustration	Stakeholder value
Evaluated_External_Evidence_SC1	Challenge_SC1_Frustration	Evaluated external evidence
Contextual_Circumstance_SC1	Challenge_SC1_Reach	Stakeholder value
Evaluated_External_Evidence_SC1	Challenge_SC1_Reach	Evaluated external evidence

Figure 72: The evidence spread using the third dashboard.

5.2 Scenario 2: Alternative solution selection

The goal of the alternative solution selection scenario is to validate if the decision design pattern can help software product managers with evidence-based alternative solution selection.

$VAL2$: To what extent can the detection of premature information using SHACL Semantic Web constraints contribute to evidence-based alternative solution selection?

A software product manager needs to select the best solution[6] to contribute to a combination of soft goals. The right solution balances the importance of soft goals. This balance results in a solution that addresses the soft goals considering their importance. The wrong solution will be unbalanced and soft goals that are less important might be addressed to a larger extent compared to soft goals of higher importance.

5.2.1 Decision ontology pattern

Table 17 presents the knowledge criteria, based on the description in section 2.5.3 Alternative solution selection, that a software product manager uses to decide which solution fits best to the defined soft goals and their importance. We define the $Soft\_Goal$, *Behavioural_Goal*, and *Score* classes as decision-relevant *root* classes. *Soft_Goal* and *Behavioural_Goal* are a sub-class of *Goal*. We need the *System* class to provide the context of the soft goals and for the reproduction of information. Figure 73 presents the alternative solution selection ontology.

[6] A solution is equal to a behavioural goal.

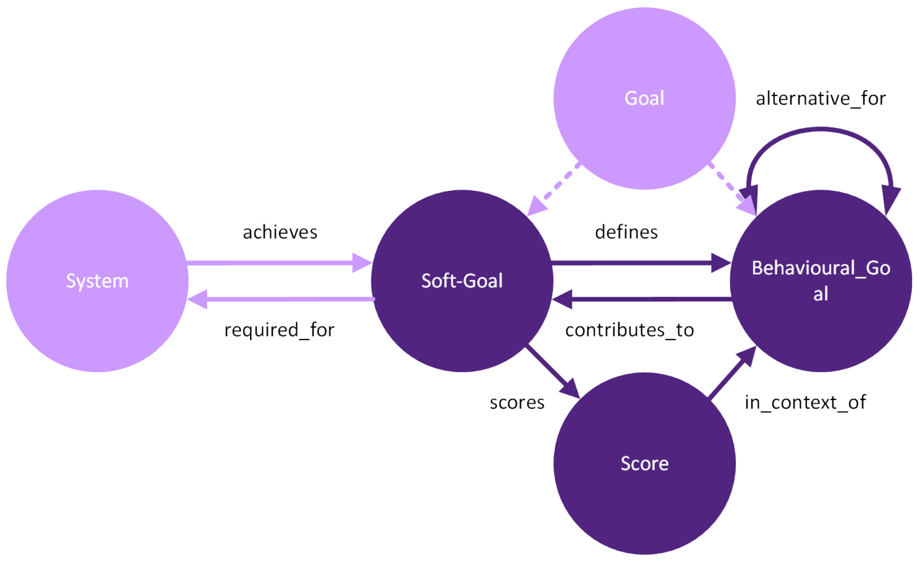


Figure 73: The alternative solution selection ontology. The violet nodes represent the decision-relevant root classes. The light violet nodes represent the classes we need for the reproduction pattern.

The *scores* and *in_context_of* object properties are essential for the structural integrity of the ontology. We use them to infer the behavioural goals that contribute to a *Soft_Goal*. A *Soft_Goal* that does not define at least one *Behavioural_Goal* leads to structural problems. Missing the *contributes_to* object property influences the reliability of the information that we present in the dashboards.

5.2.2 Instantiation of the decision ontology pattern

The instantiation of the decision ontology pattern for alternative solution selection follows the same steps as the instantiation of the decision ontology pattern for requirements prioritisation. We need to instantiate the knowledge criteria and motivate, for each knowledge criterion, the information we need to reproduce the knowledge criterion. Table 17 presents the knowledge criteria that a software product manager needs to understand to select the right solution.

We modify equation 1 in a way the definitions fit the alternative solution selection knowledge criteria. Equation 14 presents the modified equation in which bg represents an individual classified as a *Behavioural_Goal*, and sg represents an individual classified as a *Soft_Goal*.

$$totalScore(bg) = \sum_{soft\text{-}goal} (contribution(bg, sg) \times weighted\_significance(sg)) \tag{14}$$

We define the contribution and weighted significance as knowledge criteria and add them to the domain of the *Soft_Goal* and *Score* classes of the alternative solution selection ontology. We define the completeness-level of a *Soft_Goal* and *Score* based on the knowledge criteria.

Table 17: An overview of the formal knowledge criteria that alternative solution selection needs.

Data Property	Domain
soft_goal_weighted_significance	*Soft_Goal*
score_contribution	*Score*

The motivation for the reproducibility of the knowledge criteria follows a similar argumentation as we presented the validation of the requirements prioritisation scenario. We have selected the *system_objective*, *system_as_is*, *system_to_be*, and *goal_objective* as background information for the

Soft_goal_weighted_significance. Additionally, we have selected the *goal_objective* as background information for the *score_contribution*.

5.2.3 Instantiation overview

We combine the alternative solution selection ontology and the decision ontology pattern. The decision ontology pattern validates that the individuals classified as *Information* and the information types that are subclasses of *Information* are evidence-based. Figure 74 presents a conceptual overview of the instantiation. We have summarized the information types in three main information classes: *Score_∗* includes the information classes related to the score, *Goal_∗* includes the information classes related to the goal, and *System_∗* includes the information classes related to the system.

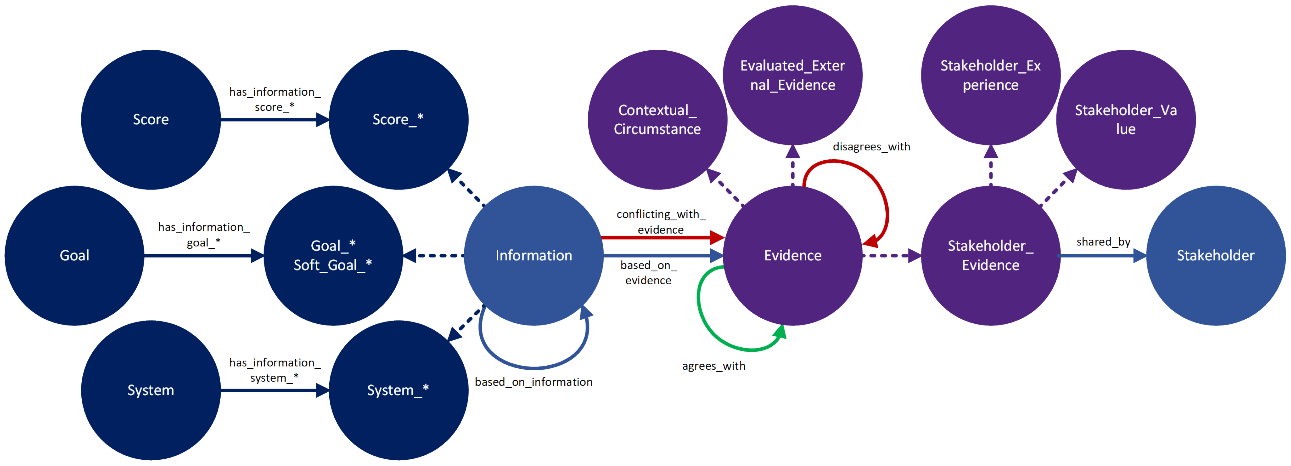


Figure 74: The combined alternative solution selection ontology and the decision design pattern ontology.

5.2.4 Structural validation

The structural validation prevents an incorrect information maturity-level. An incorrect maximum number of violations causes an incorrect information maturity-level. The structural violations are not part of the functions that define the information maturity-level. Two scenarios can cause an incorrect maximum number of violations:

1. The sum of the weighted significance of the soft goals related to one system is not equal to the total relative weight. This situation might mean a soft goal is missing, there are too many soft goals, or the values are incorrect.
2. Each behavioural goal should have at least one alternative. If this is not the case, the decision-maker has no decision to make.

If the reasoner finds structural violations, the dashboards are not reliable and will not show up. The decision-maker needs to solve the structural violations first.

Structural validation: the sum of the weighted significance

The sum of the weighted significances of the soft goals required for one system needs to be equal to an arbitrary fixed number, for example, 10. This fixed number forces the decision-maker to weigh the value of the soft goals against each other. We continue to use 10 as an example throughout the validation of the alternative solution selection scenario. Code sample 33 presents a SPARQL query embedded in a SHACL constraint[7]. The SPARQL query finds a list of systems and their total weighted significance by summing up the weighted significances of the soft goals related to the *System*. The *HAVING* statement filters out systems with a weighted significance that does not equal 10. As a result, the output of the SPARQL query lists the systems that have a weighted significance that does

[7]We removed the declared prefixes from code sample 33. The full source code is available in our GitHub repository.

not equal 10. The SHACL shape that embeds the SPARQL query generates a violation on each of those systems.

```
as:SoftGoalWeightedSignificanceShape
   sh:targetClass as:System ;
   sh:sparql [
      sh:message "Structural: ensure the weighted significance of this soft_goal equals
    to 10." ;
      sh:prefixes _:prefixes ;
      sh:select """
         SELECT $this (SUM(?dv) as ?totalws)
         WHERE
         {
               ?sg as:has_information_soft_goal_weighted_significance ?ws .
               ?ws as:data_value ?dv .
               $this as:achieves ?sg .
         }
         GROUP BY $this
         HAVING (SUM(?dv) != 10)
      """ ;
   ] .
```

Code sample 33: A SPARQL query embedded in a SHACL shape. The SHACL shape generates violations depending on the outcome of the SPARQL query.

Structural validation: behavioural goals

The software product manager needs to select the best behavioural goal that addresses the soft goals considering their weighted significance. Therefore, each $Behavioural\_Goal$ should have at least one alternative. Code sample 34 presents the SHACL shape that generates a violation if an individual that is classified as $Behavioural\_Goal$ violates this constraint.

```
as:BehaviouralGoalShape a sh:NodeShape;
   sh:targetClass as:Used_Behavioural_Goal;
   sh:property [
      sh:path as:alternative_for;
      sh:severity sh:Violation;
      sh:minCount 1;
      sh:message "Structural: add at least one alternative for this behavioural goal.";
    ].
```

Code sample 34: The SHACL shapes that generate a violation if an individual that is classified as $Behavioural\_Goal$ violates this constraint.

5.2.5 Test scenarios for the decision ontology pattern

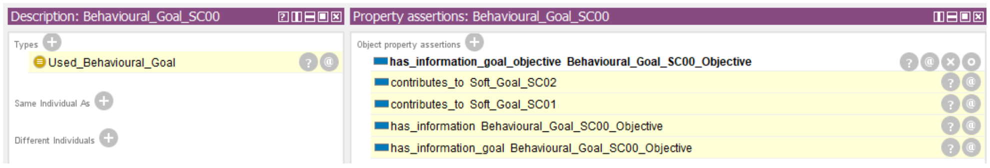
We define six abstract test scenarios. The abstraction makes it easier to validate that a specific scenario triggers the expected violations. Scenario $SC0$ validates the structure of the ontology. The first scenario ($SC1$) should not generate any violations. The other scenarios validate a specific pattern and are structurally valid. This structure limits the complexity of the scenarios and their outcomes.

Scenario $SC0$ deviates from the requirements prioritisation scenarios while we have similarly validated the other scenarios. Therefore, we describe $SC0$, including the sample data, in detail, while we summarise the results of the other scenarios in table 18.

Scenario 0 ($SC0$): Structural validation

The structural validation needs to ensure that each $Behavioural\_Goal$ has at least one alternative and that the sum of the weighted significance of the soft goals related to a system equals 10.

We create *Behavioural_Goal_SC00* without any alternatives. Figure 75 presents *Behavioural_Goal_SC00* in Protégé and confirms the lack of alternatives by not showing an *alternative_for* object property.


Description: Behavioural_Goal_SC00

Types
Used_Behavioural_Goal

Same Individual As

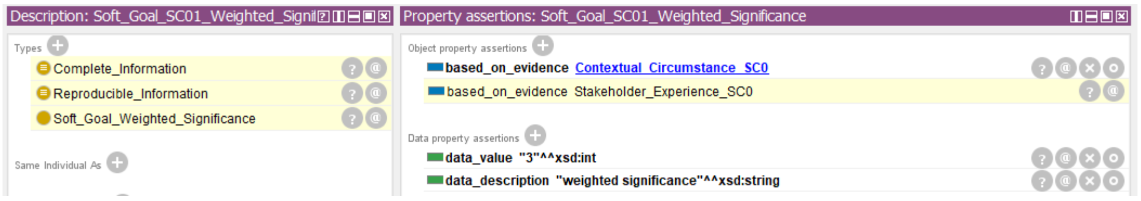
Different Individuals

Property assertions: Behavioural_Goal_SC00

Object property assertions
has_information_goal_objective Behavioural_Goal_SC00_Objective
contributes_to Soft_Goal_SC02
contributes_to Soft_Goal_SC01
has_information Behavioural_Goal_SC00_Objective
has_information_goal Behavioural_Goal_SC00_Objective

Figure 75: *Behavioural_Goal_SC00* in Protégé.

Additionally, we create *System_SC0* with two soft goals: *Soft_Goal_SC01* and *Soft_Goal_SC02*. We set the weighted significance of *Soft_Goal_SC01* to 3 and of *Soft_Goal_SC02* to 4. The sum of the weighted significance of the soft goals contributing to *System_SC0* is 7. Figures 76 and 77 present *Soft_Goal_SC01* and *Soft_Goal_SC02* in Protégé.


Description: Soft_Goal_SC01_Weighted_Signi

Types
Complete_Information
Reproducible_Information
Soft_Goal_Weighted_Significance

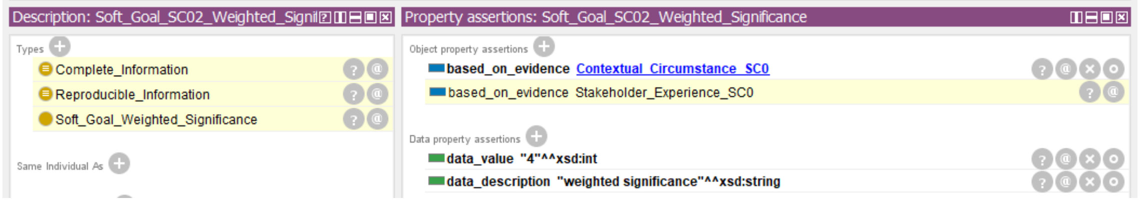
Same Individual As

Property assertions: Soft_Goal_SC01_Weighted_Significance

Object property assertions
based_on_evidence Contextual_Circumstance_SC0
based_on_evidence Stakeholder_Experience_SC0

Data property assertions
data_value "3"^^xsd:int
data_description "weighted significance"^^xsd:string

Figure 76: *Soft_Goal_SC01* in Protégé.


Description: Soft_Goal_SC02_Weighted_Signi

Types
Complete_Information
Reproducible_Information
Soft_Goal_Weighted_Significance

Same Individual As

Property assertions: Soft_Goal_SC02_Weighted_Significance

Object property assertions
based_on_evidence Contextual_Circumstance_SC0
based_on_evidence Stakeholder_Experience_SC0

Data property assertions
data_value "4"^^xsd:int
data_description "weighted significance"^^xsd:string

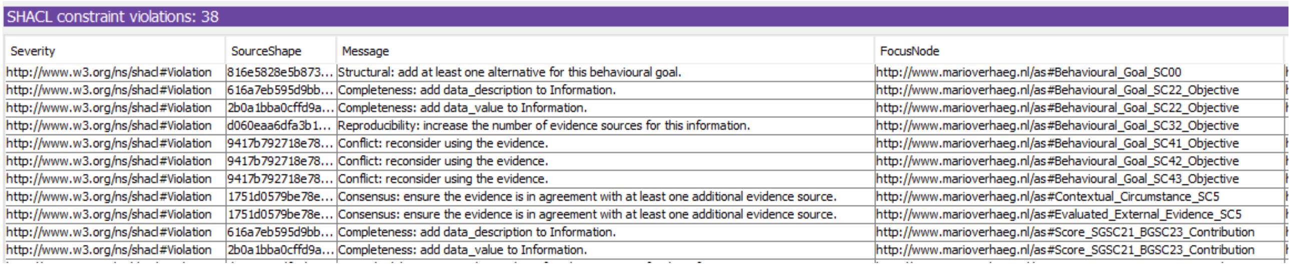
Figure 77: *Soft_Goal_SC02* in Protégé.

The result of the test scenarios

Table 18 presents the results of the tests. We detect 38 violations for the decision ontology pattern. Figure 78 presents an extract of the results as the SHACL4P plugin presents them in Protégé.

Table 18: The number of expected and detected violations per scenario and pattern.

Scenario	Scenario	Expected violations	Detected violations
SC0	Structural	2	2
SC2	n/a	0	0
SC2	Completeness	12	12
SC3	Reproducibility	4	4
SC4	Conflict	18	18
SC5	Consensus	2	2


SHACL constraint violations: 38

Severity	SourceShape	Message	FocusNode
http://www.w3.org/ns/shacl#Violation	816e5828e5b873...	Structural: add at least one alternative for this behavioural goal.	http://www.marioverhaeg.nl/as#Behavioural_Goal_SC00
http://www.w3.org/ns/shacl#Violation	616a7eb595d9bb...	Completeness: add data_description to Information.	http://www.marioverhaeg.nl/as#Behavioural_Goal_SC22_Objective
http://www.w3.org/ns/shacl#Violation	2b0a1bba0cffd9a...	Completeness: add data_value to Information.	http://www.marioverhaeg.nl/as#Behavioural_Goal_SC22_Objective
http://www.w3.org/ns/shacl#Violation	d060eaa6dfa3b1...	Reproducibility: increase the number of evidence sources for this information.	http://www.marioverhaeg.nl/as#Behavioural_Goal_SC32_Objective
http://www.w3.org/ns/shacl#Violation	9417b792718e78...	Conflict: reconsider using the evidence.	http://www.marioverhaeg.nl/as#Behavioural_Goal_SC41_Objective
http://www.w3.org/ns/shacl#Violation	9417b792718e78...	Conflict: reconsider using the evidence.	http://www.marioverhaeg.nl/as#Behavioural_Goal_SC42_Objective
http://www.w3.org/ns/shacl#Violation	9417b792718e78...	Conflict: reconsider using the evidence.	http://www.marioverhaeg.nl/as#Behavioural_Goal_SC43_Objective
http://www.w3.org/ns/shacl#Violation	1751d0579be78e...	Consensus: ensure the evidence is in agreement with at least one additional evidence source.	http://www.marioverhaeg.nl/as#Contextual_Circumstance_SC5
http://www.w3.org/ns/shacl#Violation	1751d0579be78e...	Consensus: ensure the evidence is in agreement with at least one additional evidence source.	http://www.marioverhaeg.nl/as#Evaluated_External_Evidence_SC5
http://www.w3.org/ns/shacl#Violation	616a7eb595d9bb...	Completeness: add data_description to Information.	http://www.marioverhaeg.nl/as#Score_SGSC21_BGSC23_Contribution
http://www.w3.org/ns/shacl#Violation	2b0a1bba0cffd9a...	Completeness: add data_value to Information.	http://www.marioverhaeg.nl/as#Score_SGSC21_BGSC23_Contribution

Figure 78: An extract of the results as the SHACL4P plugin presents them in Protégé.

5.2.6 Decision-relevant root individuals

We determine the decision-relevant root individuals for an alternative solution selection decision. The alternative solution selection decision typically starts with a behavioural goal bg. This behavioural goal solves a specific problem. The software product manager wants to know if there are alternatives for this behavioural goal and to what extent this behavioural goal addresses the soft goals. Code sample 35 presents a SPARQL query that retrieves the system, the soft goals, the contributing behavioural goals, and the scores using parameter $< bg >$. We execute the SPARQL query and feed the result into the set RI. We define function $asri(bg) = RI$ to logically represent the SPARQL query.

```
SELECT DISTINCT ?ri
WHERE
{
   # Gather systems
   {
      ?sg as:defines ?bg .
      ?ri as:achieves ?sg .
   }
   UNION
   # Gather soft goals
   {
      ?ri as:defines ?bg .
   }
   UNION
   # Gather scores
   {
      ?bg as:alternative_for ?bga .
      ?ri as:in_context_of ?bga .
   }
   UNION
   # Gather behavioural goals
   {
      ?bg as:alternative_for ?ri .
   }
   FILTER(?bg = as:Behavioural_Goal_SC21)
}
```

Code sample 35: A SPARQL query that retrieves the system, the required soft goals, contributing behavioural goals, and scores using parameter $< bg >$.

5.2.7 Test scenarios for the decision presentation pattern

We use one behavioural goal to validate the decision presentation pattern for alternative solution selection. The software product manager is interested in $Behavioural\_Goal\_SC21$ and wants to understand its alternatives and, eventually, select the right solution.

Decision $DEC1$**:** $Behavioural\_Goal\_SC21$

A software product manager needs to decide if $Behavioural\_Goal\_SC21$ is the right solution considering the soft goals and their significant weights. The first question we ask is:

Is the information ready for this decision?

The first dashboard of the decision presentation pattern helps a decision-maker to answer this question. We define the consolidated information maturity-level, and the evidence spread for $Behavioural\_Goal\_SC21$ to generate this dashboard.

We use function $asri(Behavioural\_Goal\_SC21)$ to define RI_{DEC1}. RI_{DEC1} is the set of decision-relevant root individuals. RI_{DEC1} includes the individuals' table 19 presents. Table 19 also presents the maximum and the actual number of violations per decision-relevant root individual. The bottom line of the table presents the maximum and the actual number of violations per pattern. We conclude that decision $DEC1$ can generate up to 97 violations. We conclude that decision $DEC1$ generates 12 completeness violations.

Table 19: The maximum number of violations per decision-relevant root individual.

Function	mvi_1	mvi_2	mvi_3	mvi_4	$Total_{mv}$	avi_1	avi_2	avi_3	avi_4	$Total_{av}$
$System\_SC2$	9	3	2	3	17	5	0	0	0	5
$Soft\_Goal\_SC21$	6	2	2	2	12	3	0	0	0	3
$Soft\_Goal\_SC22$	6	2	2	2	12	0	0	0	0	0
$Soft\_Goal\_SC23$	6	3	2	3	14	0	0	0	0	0
$Score\_SGSC21\_BGSC21$	3	1	2	1	7	0	0	0	0	0
$Score\_SGSC22\_BGSC21$	3	1	2	1	7	0	0	0	0	0
$Score\_SGSC23\_BGSC21$	3	1	2	1	7	0	0	0	0	0
$Score\_SGSC21\_BGSC22$	3	1	2	1	7	0	0	0	0	0
$Score\_SGSC22\_BGSC22$	3	1	2	1	7	0	0	0	0	0
$Score\_SGSC23\_BGSC22$	3	1	2	1	7	0	0	0	0	0
$Score\_SGSC21\_BGSC23$	3	1	2	1	7	2	0	0	0	2
$Score\_SGSC22\_BGSC23$	3	1	2	1	7	0	0	0	0	0
$Score\_SGSC23\_BGSC23$	3	1	2	1	7	0	0	0	0	0
$Behavioural\_Goal\_SC21$	3	1	2	1	7	0	0	0	0	0
$Behavioural\_Goal\_SC22$	3	1	2	1	7	2	0	0	0	2
$Behavioural\_Goal\_SC23$	3	1	2	1	7	0	0	0	0	0
Total	45	16	20	16	97	12	0	0	0	12

The information presented in table 19 allows us to calculate the information maturity-level iml for decision $DEC1$. Equation 15 shows that the information maturity-level for $DEC1$ is 88%.

$$iml(RI) = \frac{mv_(RI) - av(RI)}{mv(RI)} = \frac{97 - 12}{97} = 0.876 \tag{15}$$

The first dashboard also presents the evidence spread. Table 20 presents the evidence spread for decision $DEC1$. We use the query code sample 17 presents and add the set of decision-relevant individuals RI_{DEC1}.

Table 20: The evidence spread for the decision-relevant root individuals and the total evidence spread for this scenario.

Evidence	*Behavioural_Goal_SC21*
Contextual_Circumstance	4
Stakeholder_Value	13
Evaluated_External_Evidence	4
Stakeholder_Experience	13

Figure 79 presents a mock-up of the first decision presentation pattern dashboard. The dashboard reflects an information maturity-level of 88% and the evidence spread table 20 presents. The decision-maker needs to decide if the information maturity-level and evidence-spread are acceptable, depending on the impact of the decision.

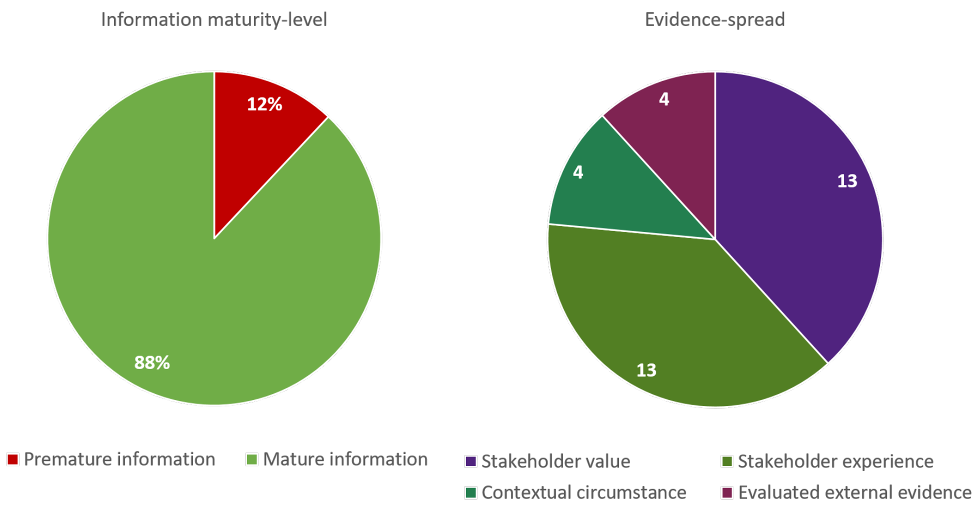


Figure 79: A mock-up of the first decision presentation pattern dashboard for test scenario $DEC1$. The dashboard reflects an information maturity-level of 0.88 and the evidence spread table 20 presents.

The second dashboard of the decision presentation pattern presents the information maturity-level per pattern. We present the pattern-specific maximum and actual violations in table 19.

The decision-relevant root individuals for $DEC1$ generate completeness violations. These violations match our expectations as scenario $SC2$ validates the completeness pattern. The completeness pattern generates 12 violations. The other patterns do not generate any violations.

Figure 80 presents a mock-up of the second decision presentation pattern dashboard. The dashboard reflects the completeness maturity-level of $\frac{45-12}{45} = 0.733$, which results in 73%. The maturity-levels of the other patterns are 100% as they do not generate any violations.

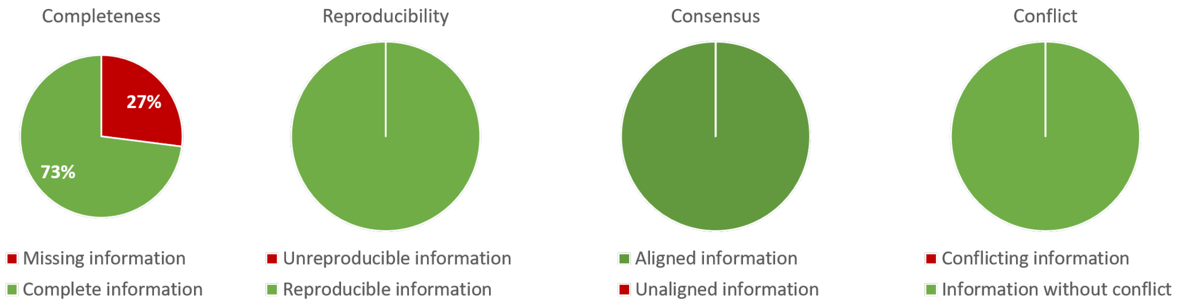
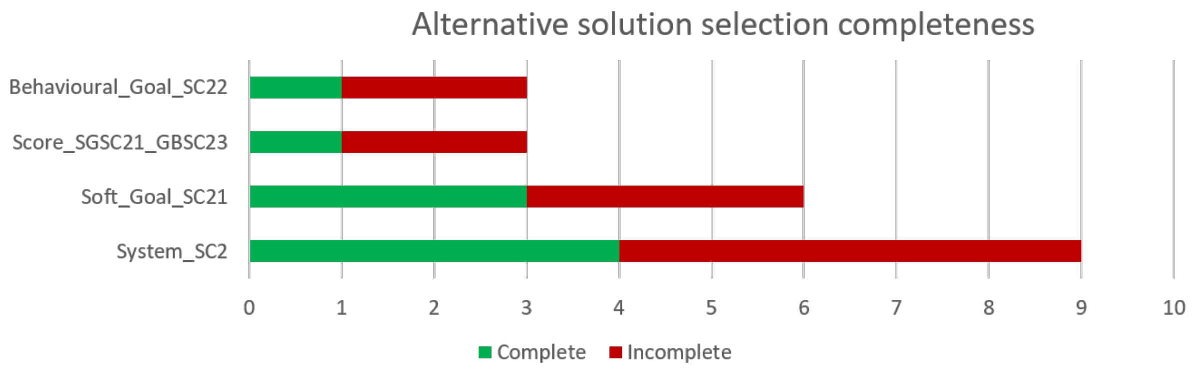


Figure 80: A mock-up of the second decision presentation pattern dashboard for decision $DEC1$.

To complete the missing information, the software product manager needs to know which information is incomplete. The third dashboard of the decision presentation pattern presents the completeness of information per decision-relevant individual.

Figure 81 presents a mock-up of the third decision presentation pattern dashboard, considering the software product manager selected $System\_SC2$. We observe that five decision-relevant individuals are incomplete. The bar chart gives the software product manager an indication which individuals are causing the 73% completeness maturity-level. The software product manager wants to know more information on $System\_SC2$. The dashboard presents related violations in the table below the bar chart. The dashboard consolidates the violations of $System\_SC2\_To\_Be$ under $System\_SC2$.



Violation	**Individual**
Completeness: add system_as_is to the System.	System_SC2
Completeness: add data_description to the system_as_is.	System_SC2
Completeness: add data_value to the system_as_is.	System_SC2
Completeness: add data_description to Information.	System_SC2_To_Be
Completeness: add data_value to Information.	System_SC2_To_Be

Figure 81: A mock-up of the third decision presentation pattern dashboard, considering the decision-maker selected $System\_SC2$. We observe that five decision-relevant individuals are incomplete.

6 Conclusion

Compared to the methodology, we work through the questions the other way around. We first address the five sub-questions related to the ontology structure and data presentation. Then we address the main research question and discuss what we have learned.

6.1 Summary of our results

This section provides a summary of our results based on the research questions presented in section 3.1 Research question.

Ontology structure

$DS1$: To what extent can we create a generic ontology design pattern for evidence-based management?

The evidence-based management pattern stores four evidence types that we describe in section 2.1.1 Decision-making model: contextual circumstances, evaluated external evidence, stakeholder values, and stakeholder experience. The evidence-based management pattern is a generic ontology design pattern. Domain experts can easily instantiate this pattern into an evidence-based management ontology to extend evidence-based management into their (existing) ontology. In our context, the evidence-based management pattern provides the decision ontology pattern with evidence storage and classification capabilities. Additionally, the decision presentation pattern calculates the evidence-spread using the evidence classification of the evidence-based management pattern.

$DS2$: To what extent can we store the information completeness and reliability in an (extended) evidence-based management ontology?

We define premature information as a combination of information completeness and information reliability. Additionally, we define information reliability as a combination of reproducibility, conflict, and consensus. The evidence-based management ontology stores the evidence that serves as the base of decision-relevant information. We extend the evidence-based management ontology with the decision ontology pattern. We instantiate the decision ontology pattern into the decision specific ontology that stores decision-relevant information. The decision ontology relates decision-relevant information to evidence. The decision ontology pattern embeds the completeness, reproducibility, consensus, and conflict patterns. When instantiated, these patterns detect premature information using Semantic Web inferencing, consistency, and constraints.

$DS3$: To what extent can we use Semantic Web inferencing, consistency, and constraints to detect premature information?

The decision ontology pattern uses Semantic Web inferencing to generate new object properties and classifies individuals. Inferencing shortens the path between two individuals by inferring a new object property based on a chain of existing object properties. The shortened path allows us to reduce the complexity of the constraints. Additionally, inferencing classifies individuals based on their properties. The classification ensures the individuals are validated using the right constraints.

A reasoner infers object properties and classifies individuals based on complex characteristics, for example, transitivity or a chain of object properties. Semantic Web consistency prevents structural (human) mistakes in the structure of the ontology and, therefore, prevents the decision-maker from spending time on understanding inconsistent information.

The decision ontology pattern relates decision-relevant information to evidence. This relationship allows the completeness, reproducibility, consensus, and conflict patterns to detect premature information using SHACL cardinality constraints. The minimum cardinality detects incomplete information, unreproducible information, and evidence that does not meet the consensus requirements. The maximum cardinality detects conflicting evidence. We introduce parameters into the Semantic Web constraints to make them re-usable.

Data presentation

$PRES1$: How do we define the maturity-level of decision-relevant information?

The information maturity-level represents mature and premature information. For example, if a decision can generate up to 100 violations while it generates 30 violations, the information maturity-level is 70%. The information maturity-level is 0% if the decision generates 100 violations. This number, typically presented using a graph or chart, explains the decision-maker the ratio between premature and mature information.

$PRES2$: Under which circumstances is a decision-maker interested in the maturity-level of decision-relevant information?

Decisions have an impact on an organisation or environment. The impact of a decision can be positive and negative. For example, a decision can result in an increase (positive) or decrease (negative) of revenue. We assume the decision-maker wants to prevent a negative impact. The information maturity-level allows the decision-maker to understand the completeness and reliability of the decision-relevant information. For example, 50% of the required information is missing and, the 50% that is complete cannot be reproduced or is in conflict with other information. This situation indicates that the impact of the decision is not clear. The limited understanding of the impact of a decision can result in a negative impact, which is something the decision-maker wants to prevent.

$PRES3$: To what extent can we make it easier for a decision-maker to understand the maturity-level of decision-relevant information?

The decision presentation pattern presents the information maturity-level using three dashboards. The first dashboard presents the decision specific information maturity-level and evidence spread. The second dashboard presents the completeness, reproducibility, consensus, and conflict maturity-levels and allows the decision-maker to understand how the decision specific information maturity-level is calculated. The third dashboard presents the premature information that the decision-maker needs to address to increase the information maturity-level. The first dashboard allows a decision-maker to understand the current status of the information maturity-level quickly. When the decision-maker finds the information maturity-level too low, the decision-maker can use the second and third dashboard to define actions to increase the information maturity-level.

6.2 Main research question

RQ: To what extent can the detection of premature information using SHACL Semantic Web constraints contribute to evidence-based decision-making?

The decision design pattern consolidates five generic ontology design patterns and a data presentation pattern. The evidence-based management pattern is a generic ontology design pattern that stores and classifies evidence. The decision ontology pattern uses four generic ontology design patterns: completeness, reproducibility, consensus, and conflict. These patterns use Semantic Web constraints to detect premature information. The decision presentation pattern calculates the information maturity-level using the detected premature information. The presentation of the information maturity-level gives the decision-maker an understanding of the decision-relevant information completeness and reliability. The decision-maker can use this understanding to increase the information maturity-level. The decision-maker can increase the information maturity-level by increasing the completeness, reproducibility, and consensus while decreasing the conflict.

A transparent information maturity-level allows the decision-maker to increase the information maturity-level, which contributes to evidence-based decision-making.

6.3 Discussion

What if a decision-maker wants to start using our work tomorrow to start analysing the decision-relevant information quality?

Practical considerations

The decision design pattern provides an information structure wrapped in an ontology. It provides mechanisms to extract the information maturity-level from this structure. However, the decision presentation pattern does not include a way to transform the numeric values representing the information maturity-level into a user-interface. The graphs and charts we present are mock-ups. Additionally, the tools we use to manipulate and validate the information, for example, Protégé and HETS, are not suitable for day-to-day usage. These tools require knowledge that a typical decision-maker or domain expert does not poses, for example, using a SPARQL query or a SHACL shape. This lack of knowledge can be easily mitigated by involving the right experts. Researches are still working on the extension that allows HETS to validate and instantiate GDOL. As a result, the instantiation of the generic ontology design patterns is not straightforward yet and requires the development of new tools or manual labour.

Impact of the results

In this paragraph, we set aside the practical considerations and assume the decision-maker can use our work tomorrow. Imagine a situation in which two stakeholders are trying to convince a decision-maker to make a decision based on their argumentation and preference. The arguments both stakeholders bring into the decision are valid based on the information they present. However, the decision-maker does not know if this information can be trusted. The decision-maker can evaluate the information maturity-level if the organisation has implemented the decision design pattern. This analysis allows the decision-maker to understand the completeness, reproducibility, consensus, and conflict-related to the information the stakeholders present. The decision-maker can do two things using this knowledge:

1. Make a decision based on the combination of the available argumentation and the information maturity-level.
2. Decline to make a decision and instruct the stakeholders to improve their information maturity-level and, as a consequence, update their argumentation.

Without knowing the information maturity-level, the argumentation brought by *Stakeholder_A* might be more persuasive compared to the argumentation brought by *Stakeholder_B*. The argumentation might have convinced the decision-maker to decide in favour of *Stakeholder_A*. However, if the information *Stakeholder_A* used for the argumentation is not reproducible or contains multiple conflicts, the decision might cause an unexpected outcome.

Learnings

We learned that the combination of inferencing, consistency, and constraints, embedded in Semantic Web technologies, can be quite useful in the context of decision-making. Decision-makers base decision on some form of information and evidence. Semantic Web technologies can provide structure to decision-relevant information, infer new information, and detect information quality issues. Decision-makers can use the structural and inferencing capabilities Semantic Web technologies introduce to increase their understanding of the decision-relevant information. Additionally, Semantic Web technologies can detect information quality issues early that allows the decision-maker to complete missing information, ensure information is evidence-based and reproducible, increase the consensus of information, and correct conflicting information.

Recommendation for future research

This section lists, based on our experience, the most relevant opportunities for future research. Solving the practical challenges would increase the applicability of our study.

We manually process the output of the Semantic Web constraints, calculate the information maturity-level, and create mock-ups to present the information maturity-level. We suggest that a future study can build bridges between the violations that the Semantic Web constraints generate, the functions that calculate the information maturity-level, and the decision presentation pattern that presents the information maturity-level. This subject has opportunities for generalisation as well by, for example, defining a pattern for presenting constraint violations. Additionally, we instantiated re-usable SHACL constraints manually in this study. Generic constraint patterns could take SHACL shapes and instantiate them using parameters. Generic constraint design patterns could make it easier for domain experts to re-use existing SHACL shapes.

The scope of this study excluded the manipulation of decision-relevant information. Decision-makers need to be able to manipulate the information to increase the information maturity-level. Several other studies at the Open University have looked at the usage of Fresnel forms in the context of the Semantic Web. It might be interesting to evaluate if a similar approach is suitable to extend the decision design pattern with, for example, a decision-relevant information manipulation pattern.

Our last suggestion extends the usefulness of Semantic Web constraints, and specifically SHACL. We use Semantic Web constraints to validate the information completeness, reproducibility, consensus, and conflict of decision-relevant information. However, knowing the information completeness, reproducibility, consensus, and conflict might be useful in other scenarios as well. Compared to our study, the researchers should focus on Semantic Web constraints, while they extend the scope of the scenarios outside of the decision-making domain. This study could measure the quality of the information an ontology stores using Semantic Web constraints.

References

Maglyas, A., Nikula, U., Smolander, K., & Fricker, S. A. (2017). Core software product management activities. *Journal of Advances in Management Research*, *14*(1), 23–45 (cit. on pp. 5, 13).

Büyüközkan, G., & Feyzıoglu, O. (2004). A fuzzy-logic-based decision-making approach for new product development. *International journal of production economics*, *90*(1), 27–45 (cit. on p. 5).

Saltan, A., Jansen, S., & Smolander, K. (2018). Decision-making in software product management: Identifying research directions from practice. *2018*, 164–176 (cit. on pp. 5, 8).

Mind the Product. (2019). *Mind the product*. https://www.mindtheproduct.com/category/data-driven-product-management/. (Cit. on p. 5)

Baba, V. V., & HakemZadeh, F. (2012). Toward a theory of evidence-based decision making. *Management decision*, *50*(5), 832–867 (cit. on pp. 5, 8, 25, 41).

Nicolas, R. (2004). Knowledge management impacts on decision making process. *Journal of knowledge management*, *8*(1), 20–31 (cit. on pp. 5, 8).

Shadbolt, N., Berners-Lee, T., & Hall, W. (2006). The semantic web revisited. *IEEE intelligent systems*, *21*(3), 96–101 (cit. on pp. 6, 8).

Harker, P. T. (1989). The art and science of decision making: The analytic hierarchy process. In *The analytic hierarchy process* (pp. 3–36). Springer. (Cit. on p. 7).

Briner, R. B., Denyer, D., & Rousseau, D. M. (2009). Evidence-based management: Concept cleanup time? *Academy of Management Perspectives*, *23*(4), 19–32 (cit. on pp. 7, 8, 41).

Johns, G. (2006). The essential impact of context on organizational behavior. *Academy of management review*, *31*(2), 386–408 (cit. on p. 8).

Brickley, D., Guha, R. V., & McBride, B. (2015). *RDF schema 1.1*. https://www.w3.org/TR/rdf-schema/. (Cit. on p. 8)

Prud'Hommeaux, E., Seaborne, A. et al. (2008). *SPARQL query language for RDF*. https://www.w3.org/TR/rdf-sparql-query/. (Cit. on p. 8)

Knublauch, H. (2011). *SPIN - modeling vocabulary*. https://www.w3.org/Submission/spin-modeling/. (Cit. on p. 8)

Knublauch, H., Hendler, J. A., & Idehen, K. (2011). Spin-overview and motivation. *W3C Member Submission*, *22*, W3C (cit. on p. 8).

Welty, C., McGuinness, D. L., & Smith, M. K. (2004). *OWL web ontology language guide*. https://www.w3.org/TR/owl-guide/. (Cit. on p. 8)

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, *5*(2), 51–53 (cit. on p. 9).

Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A., et al. (2004). *OWL web ontology language reference*. https://www.w3.org/TR/owl-ref/. (Cit. on pp. 9, 38)

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., Dean, M., et al. (2004). Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, *21*(79), 1–31 (cit. on p. 9).

Boneva, I., Gayo, J. E. L., & Prud'hommeaux, E. G. (2017). Semantics and validation of shapes schemas for RDF. In *International semantic web conference*. Springer. (Cit. on p. 9).

Knublauch, H., & Kontokostas, D. (2017). *Shapes constraint language (SHACL)*. https://www.w3.org/TR/shacl/. (Cit. on pp. 9, 10)

Spahiu, B., Maurino, A., & Palmonari, M. (2018). Towards improving the quality of knowledge graphs with data-driven ontology patterns and SHACL. *Emerging Topics in Semantic Technologies: ISWC 2018 Satellite Events*, *36*, 103 (cit. on p. 9).

Li, T., Feng, S., & Li, L. X. (2001). Information visualization for intelligent decision support systems. *Knowledge-Based Systems*, *14*(5-6), 259–262 (cit. on p. 10).

Al-Kassab, J., Ouertani, Z. M., Schiuma, G., & Neely, A. (2014). Information visualization to support management decisions. *International Journal of Information Technology & Decision Making*, *13*(02), 407–428 (cit. on p. 10).

Coury, B. G., & Boulette, M. D. (1992). Time stress and the processing of visual displays. *Human factors*, *34*(6), 707–725 (cit. on p. 10).

Coles, S. (1997). Creating effective graphs and charts. *Industrial management & data systems* (cit. on pp. 10, 11, 46).

Hardin, M., Hom, D., Perez, R., & Williams, L. (2012). Which chart or graph is right for you? (Cit. on pp. 10, 13, 46–48).

Kleiner, F. (2015). *A semantic wiki-based platform for IT service management*. KIT Scientific Publishing. (Cit. on p. 11).

Devedzić, V. (2002). Understanding ontological engineering. *Communications of the ACM*, *45*(4), 136–144 (cit. on p. 11).

Soshnikov, D. (2003). Ontological design patterns in distributed frame hierarchy. In *Proceedings of the 5th international workshop on computer science and information technologies, Ufa, Russia*. Citeseer. (Cit. on pp. 11, 12).

Krieg-Brückner, B., Mossakowski, T., & Neuhaus, F. (2019). Generic ontology design patterns at work. *arXiv preprint arXiv:1906.08724* (cit. on p. 12).

Svatek, V. (2004). Design patterns for semantic web ontologies: Motivation and discussion. In *Proceedings of the 7th conference on business information systems, poznan*. (Cit. on p. 12).

Krieg-Brückner, B., & Mossakowski, T. (2017). Generic ontologies and generic ontology design patterns. In *Wop@ iswc*. (Cit. on p. 12).

Regnell, B., Höst, M., Natt och Dag, J., Beremark, P., & Hjelm, T. (2000). Visualization of agreement and satisfaction in distributed prioritization of market requirements (cit. on p. 13).

Matsokis, A., & Kiritsis, D. (2010). An ontology-based approach for product lifecycle management. *Computers in industry*, *61*(8), 787–797 (cit. on p. 13).

Ameri, F., & Dutta, D. (2005). Product lifecycle management: Closing the knowledge loops. *Computer-Aided Design and Applications*, *2*(5), 577–590 (cit. on p. 13).

Weerd, I., Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., & Bijlsma, A. (2006). A reference framework for software product management. *Technical report UU-CS*, *2006*(014) (cit. on p. 13).

Natt och Dag, J., Regnell, B., Gervasi, V., & Brinkkemper, S. (2005). A linguistic-engineering approach to large-scale requirements management. *IEEE software*, *22*(1), 32–39 (cit. on p. 13).

Maglyas, A., Nikula, U., & Smolander, K. (2012). Lean solutions to software product management problems. *IEEE software*, *29*(5), 40–46 (cit. on pp. 13, 15).

Mind the Product Training. (2018). Essentials 102, increasing your strategic impact as a product manager. (Cit. on pp. 13, 14).

Van Lamsweerde, A. (2009). *Requirements engineering: From system goals to UML models to software* (Vol. 10). Chichester, UK: John Wiley & Sons. (Cit. on pp. 14–17).

Berander, P., & Andrews, A. (2005). Requirements prioritization. In *Engineering and managing software requirements* (pp. 69–94). Springer. (Cit. on p. 14).

Lehtola, L., Kauppinen, M., & Kujala, S. (2004). Requirements prioritization challenges in practice. In *International conference on product focused software process improvement*. Springer. (Cit. on p. 15).

Wiegers, K. (1999). First things first: Prioritizing requirements. *Software Development*, *7*(9), 48–53 (cit. on p. 15).

Achimugu, P., Selamat, A., Ibrahim, R., & Mahrin, M. N. (2014). A systematic literature review of software requirements prioritization research. *Information and software technology*, *56*(6), 568–585 (cit. on p. 15).

Aurum, A., & Wohlin, C. (2003). The fundamental nature of requirements engineering activities as a decision-making process. *Information and Software Technology*, *45*(14), 945–954 (cit. on p. 15).

van Lamsweerde, A. (2009). Reasoning about alternative requirements options. In *Conceptual modeling: Foundations and applications* (pp. 380–397). Springer. (Cit. on p. 16).

Lapouchnian, A. (2005). Goal-oriented requirements engineering: An overview of the current research. (Cit. on p. 16).

Mylopoulos, J., Chung, L., & Nixon, B. (1992). Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on software engineering*, *18*(6), 483–497 (cit. on p. 16).

de Klerk, S. (2018). *Semantic web content ontology design patterns for A-Hohfeld legal relations* (Master's thesis). Open Universiteit Nederland. (Cit. on p. 20).

Saunders, M., Lewis, P., & Thornhill, A. (2015). *Methoden en technieken van onderzoek. 7de dr.* Amsterdam: Pearson. (Cit. on p. 20).

Musen, M. A. et al. (2015). The Protégé project: A look back and a look forward. *AI matters*, *1*(4), 4 (cit. on p. 21).

Holger Knublauch. (2017). *From SPIN to SHACL*. https://spinrdf.org/spin-shacl.html. (Cit. on p. 22)

Ekaputra, F. J., & Lin, X. (2016). SHACL4P: SHACL constraints validation within Protégé ontology editor. In *2016 international conference on data and software engineering (icodse)*. IEEE. (Cit. on p. 22).

Noy, N., & Rector, A. (2006). *Defining n-ary relations on the semantic web*. https://www.w3.org/TR/swbp-n-aryRelations. (Cit. on pp. 37, 38)

Stone, M. (2006). Choosing colors for data visualization. *Business Intelligence Network*, *2* (cit. on p. 46).