2021

Towards a Generic Method for Articulating Design Uncertainty

Mouna Dhaouadi
Université de Montréal

Kate M. B. Spencer
Smith College

Megan H. Varnum
Smith College

Alicia M. Grubb
Smith College, amgrubb@smith.edu

Michalis Famelis
Université de Montréal

JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

Towards a Generic Method for Articulating Design-time Uncertainty

Mouna Dhaouadi[*], **Kate M. B. Spencer**[†], **Megan H. Varnum**[†], **Alicia M. Grubb**[†], **and Michalis Famelis**[*]
[*]Université de Montréal, QC, Canada
[†]Smith College, MA, USA

ABSTRACT Modelers encounter different kinds of uncertainty in their designs and models of software systems. One such type concerns uncertainty about how to build a model. This is called design-time uncertainty, and existing research has studied how modelers can work in its presence. However, the process by which they come to elicit and express their uncertainties remains unclear. In this paper, we take steps towards addressing this gap by introducing DRUIDE (Design and Requirements Uncertainty Integrated Development Environment), a language and workflow for articulating design time uncertainty. We present and illustrate our proposal on a software design example. Additionally, we conduct a real life case study of domain analysis related to the uncertainty caused by the COVID-19 pandemic, and evaluate DRUIDE with it. Our evaluation shows that DRUIDE is sufficiently expressive to articulate design time uncertainty.

KEYWORDS Uncertainty language, modeling language, model-driven engineering, requirements models, design-time decisions.

1. Introduction

The requirements and specification of software systems continue to evolve during the modeling and design phase, as additional information is gathered from stakeholders. Making decisions at the right time is key to successfully performing engineering tasks. Making decisions without all necessary information may lead to wasted effort or to premature commitment to potentially dangerous assumptions; however, modelers are often required to work in the presence of uncertainty. One way to handle such contingency is to avoid uncertain parts of the system for as long as possible (Poppendieck & Poppendieck 2003); however, this may lead to under-utilization of resources. Another approach is to explicitly articulate uncertainty in models and to treat uncertainty as a first-class concern in model-based decision making (Walker et al. 2003).

Consider a team of modelers who want to create a tool for modeling concurrent systems with place/transition nets (PTNs), such as the model in Fig. 1(a). They create the metamodel for PTNs, shown in the dashed box in Fig. 1(b). Its entry point is the class Net that contains all other classes. Then they create the class Transition to represent the events that can occur in a PTN model, represented by the box element 3 in the instance model in Fig. 1(a). The modelers create the Place class to represent the conditions that need to be met in order for such an event to happen, represented by the circle elements 1 and 5. The Token class indicates which conditions are satisfied at a given point in time, using a black marker token as shown for place 1 in Fig. 1(a). Finally, to represent the connections between places and transitions (i.e., the arrows labeled 2 and 4), they add the PlaceToTransitionArc and TransitionToPlaceArc classes to the metamodel in Fig. 1(b).

As the requirements of their tool evolve, the team needs to adapt their metamodel and it is not obvious how to proceed. For example, should the arcs between places and transitions be reified as separate classes? The same effect could be modelled in a more memory-efficient way as associations between the Place and Transition classes. They also consider the possibility of extending their metamodel to support events prioritization by adding weights to the arcs, which requires them to be reified. Finally, they are unsure if the metamodel should allow PTN elements to store the location of their graphical representations

JOT reference format:
Mouna Dhaouadi, Kate M. B. Spencer, Megan H. Varnum, Alicia M. Grubb, and Michalis Famelis. *Towards a Generic Method for Articulating Design-time Uncertainty*. Journal of Object Technology. Vol. 21, No. 3, 2021.
http://dx.doi.org/10.5381/jot.2021.21.3.a3

in PTN diagrams. Doing so makes it easier to store human-readable diagrams but makes the metamodel overly coupled with the concrete syntax.

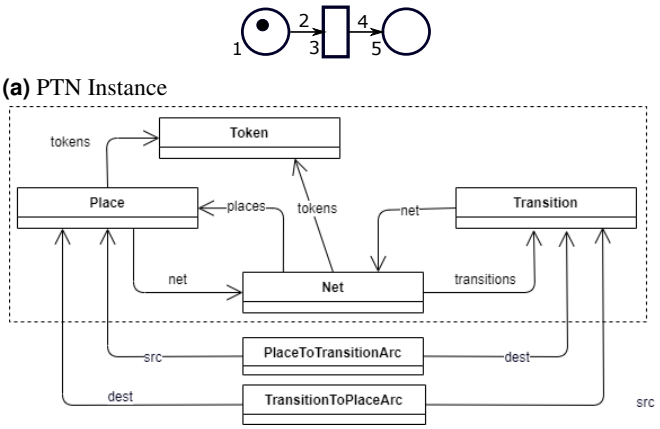
Making these decisions without enough information is risky, as the team may have to undo some of their work if they make the wrong assumption. But avoiding these decisions is also risky, as designs chosen provisionally in the present can become constraints for decisions in the future. Worse, the team may lose the context for why certain designs were chosen, as well as which designs were provisional and which were the result of careful consideration. These problems may result in costly reinventions of the same solution.

In summary, the team members face uncertainty about how to evolve the design of their metamodel, due to the lack of information about the future needs of their system. They need to keep track of the points of uncertainty, and the different alternative design scenarios, with their various degrees of interdependency. Modelers should be able to:

MC1 express that they are uncertain about how to design some aspects of their model;
MC2 be as vague as necessary at first, while they explore possible solutions;
MC3 indicate what parts of the model they are uncertain about;
MC4 elaborate their uncertainty into concrete design decisions, as they understand the implications of each design;
MC5 express the evolution of these design decisions;
MC6 describe dependencies between the design decisions; and
MC7 show how to make decisions actionable in the model.

These types of uncertainty concern the modelers' lack of information about how to design a model. It is known as *design time uncertainty* and can affect models at any point in the software lifecycle. Design time uncertainty can be made explicit and managed in models (Famelis & Chechik 2019). Various aspects of working with design time uncertainty have been studied in the literature, such as using partial models for reasoning (Famelis et al. 2012). Other approaches for modeling design time uncertainties have been developed in the context of bidirectional transformations using JTL (Eramo et al. 2015), architectural interfaces (Watanabe et al. 2017), and pattern matching (Semeráth & Varró 2017). All these approaches rely on articulating the uncertainty that the modeler has about design decisions directly in a software model. To the best of our knowledge, previous work takes for granted that modelers have already expressed their uncertainty in a model. In other words, the process by which this is elicited and expressed remains unclear.

In this paper, we propose a generic language and approach that supports articulating and evolving design time uncertainty, called DRUIDE (Design and Requirements Integrated Development Environment). We llustrate its usage and evaluate it using a case study from the COVID-19 pandemic. We pose the following research question (RQ): *"To what extent can* DRUIDE *be used to articulate uncertainty?"*. We consider uncertainty within the same model (i.e., intra-model) or across related models (i.e., inter-model) in our in-depth case study. We used the case study to evaluate the adequacy and expressiveness of DRUIDE, as well as to draw wider conclusions about modeling heterogeneous types of uncertainty. We envision the



(a) PTN Instance

(b) PTN Metamodel

Figure 1 Modeling Place-Transition Nets (PTNs)

following benefits of DRUIDE: (a) it allows the "separation of modeling concerns" regarding different types of uncertainty (see Sect. 5); (b) it improves the state of the art in design-time uncertainty management allowing better leveraging of existing techniques (see Sect. 7); and (c) it is a step towards lowering the barrier to adoption for some design rationale modeling techniques (see Sect. 7).

In this paper, we make three contributions: (1) a language for modelers to express design time uncertainty; (2) a workflow for using the language to express and localize design uncertainty, as well as evolve and operationalize design decisions; and (3) an in-depth real world evaluative case study of uncertainty that spans heterogeneous models and their relationships.

The remainder of this paper is organized as follows: We introduce background work in Sect. 2 and present DRUIDE in Sect. 3. We illustrate and evaluate DRUIDE with a case study in Sect. 4, and discuss these results in Sect. 5. We discuss threats in Sect. 6, related work in Sect. 7, and conclude in Sect. 8.

2. Background

In this section, we present relevant background work, focusing on concepts that are influential to the development of DRUIDE.

Walker et al. defined a harmonized terminology and typology of uncertainty in model-based decision support (Walker et al. 2003). They provided a conceptual framework for the systematic treatment of uncertainty, from a modeler's perspective, in order to improve its management in decision making processes. Walker et al. suggested that uncertainty is a three dimensional concept, defined by a matrix, consisting of its nature, level, and location. We use these concepts to characterize different types of uncertainty found in a model-based system. Walker et al. construed the *level of uncertainty* as a continuous progression between determinism and ignorance, classifying six levels: (1) *Determinism*, where everything is known precisely; (2) *Statistical Uncertainty*, where uncertainty follows a known statistical distribution and is thus predictable; (3) *Scenario Uncertainty*, where there exists of a set of possible alternative scenarios, and the uncertainty resides in not knowing which one to pick; (4) *Recognized Ignorance*, where modelers acknowledge that they don't know something (i.e., known unknowns); (5) *Indetermi-*

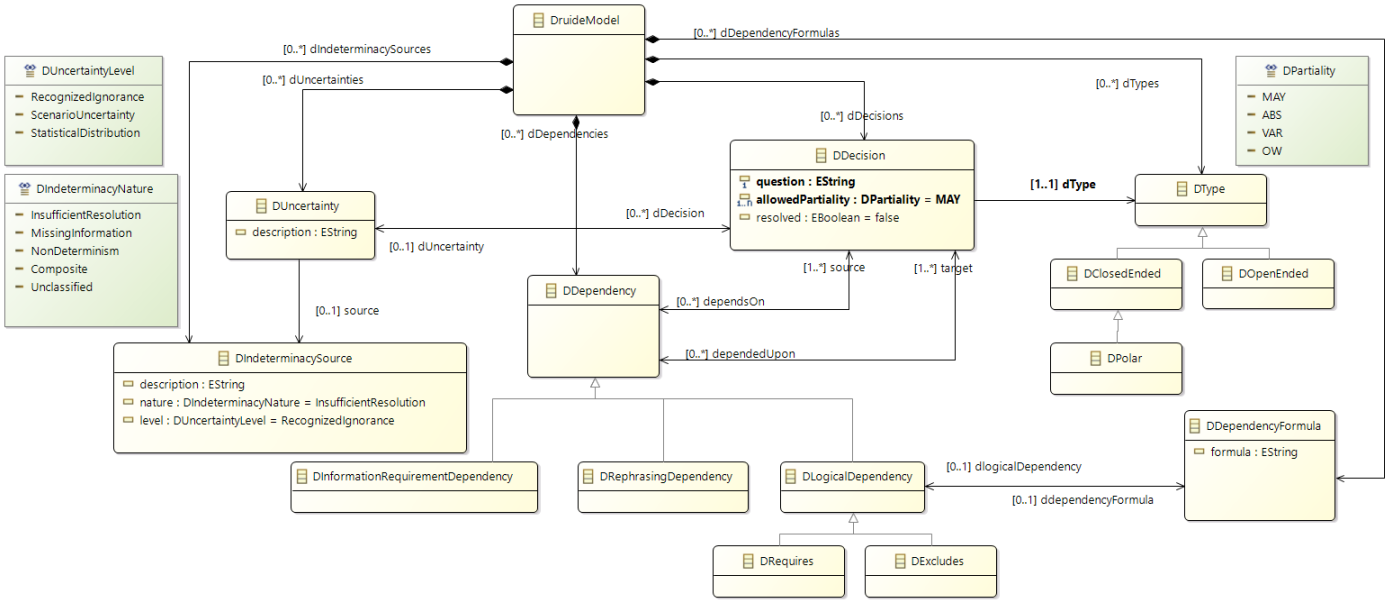


Figure 2 DRUIDE Metamodel

nacy (also called *Irreducible ignorance*), where it is impossible to resolve the ignorance due to its indeterminate nature (i.e., at the edge of *Recognized Ignorance*); and (6) *Total Ignorance*, which represents a higher order of uncertainty, where modelers are not even aware that they do not have enough information (i.e., unknown unknowns).

Zhang et al. proposed the U-Model: a conceptual model for uncertainty specifically designed for Cyber-Physical Systems (CPSs) (Zhang et al. 2016). They extended the Restricted Use Case modeling (RUCM) methodology (Yue et al. 2013), to identify and specify uncertainty as part of system requirements (Zhang et al. 2018), resulting in the U-RUCM methodology. They showed the efficiency of their method based on two case studies of industrial Cyber-Physical Systems. The U-Model includes a *BeliefModel*, a *MeasureModel* and an *UncertaintyModel* and is mapped to the CPSs three logical levels: Application, Infrastructure, and Integration.

The U-Model introduced uncertainty related concepts, such as *Indeterminacy*, *IndeterminacySource* and *IndeterminacyNature*. *Indeterminacy* refers to a situation where necessary knowledge is unavailable. *IndeterminacySource* represents the factors that lead to the uncertainty of an *Indeterminacy* element. As there are several kinds of *IndeterminacySources*, Zhang et al. categorized them with different *IndeterminacyNatures*. We list the categories of *IndeterminacyNature* and their explanations: (a) *Insufficient Resolution*: the available information is not precise enough. (b) *Missing Information*: some related information is unavailable. (c) *Non-determinism*: the phenomenon is non-deterministic. (d) *Composite*: a combination of two or more indeterminacy natures. (e) *Unclassified*: none of the above. For example, consider a modeler exploring the usage of multiple inheritance in a high-level design without knowing if the programming language used for implementation supports this option, due to an incomplete specification. The *Indeterminacy* is the uncertainty about whether multiple inheritance should be used or not. Its *IndeterminacySource* is the incomplete specification and its associated *IndeterminacyNature* is *MissingInformation*.

We are interested in uncertainty that concerns design decisions. The Decision Requirements Diagram (DRD), part of OMG's Decision Model and Notation (DMN) (OMG 2020-03), is composed of *elements* that constitute the domain of decision making, and the dependencies between them. Dependencies between elements represent three types of requirements: *Knowledge Requirement*, *Authority Requirement* and *Information Requirement*. The *Information Requirement* captures the idea that an output of a decision is used as input to another decision.

Decision modeling is also used for Software Product Line Engineering (SPLE). In the context of SPLE, DOPLER (Decision-Oriented Product Line Engineering for effective Reuse) is the best known variability modeling approach (Dhungana et al. 2007). The DoplerVML modeling language (Dhungana et al. 2011), distinguishes between *Decisions* and *Assets*. Decisions are used to represent the problem space (i.e., the available customization options), while Assets are used to define the solution space (i.e., the parts required to compose the product). The model also defines traceability links between Decisions and Assets, which enables a specific product configuration to be generated from a customer's input for the decisions. Schmid et al. reviewed several decision modeling approaches for product lines, including DOPLER, and defined their common elements as a basic model structure (Schmid et al. 2011). Schmid et al. observed that all approaches define a *Question* attribute, as part of the Decision element, that describes the decision to the user. The approaches agree that there exists different types of decisions, supporting Boolean and Enumerated types. Additionally, the approaches allow for creating dependencies between decisions and a hierarchy of dependency types.

Famelis and Chechik proposed managing the lifecycle of design time uncertainty using partial models (Famelis & Chechik 2019), based on a simplification of MAVO (Salay et al. 2012).

Constraint	Rationale
A *DDependency* element cannot *dependsOn* and *dependedUpon* the same *DDecision*.	Avoid self dependencies.
A set of *DDecisions* cannot be connected by *DDependency* elements to form a cycle.	Avoid cycle dependencies.
A pair (or set) of *DDecisions* cannot have more than one *DDependency* element between them.	DRUIDE introduces a hierarchy of dependencies that is intended to capture all possible dependency types between a set of decisions.
A set of only *DPolar* decisions cannot be the *source* of a *DRephrasingDependency*.	*DPolar* decisions (i.e., binary questions) should be answered by yes or no. We consider them to be the final result of the thinking process, where a decision has been reduced to a clear choice.
DLogicalDependencies can only exist between sets of *DPolar* decisions.	Only *DPolar* decisions can be atomically expressed as logical propositions.
The *source* and *target* decisions of a *DRephrasingDependency* must have the same *dUncertainty*.	When a set of decisions rephrases another set of decisions, they should all concern the same uncertainty.

Table 1 Well-formedness constraints for *DDependency* links between *DDecision* elements.

MAVO is a formal approach for modeling partiality in a metamodel-agnostic way, and distinguishes four partiality types: *May*, *Abs*, *Var*, and *OW*. *May* partiality concerns uncertainty about the inclusion of a particular element in the model. In Famelis et al. (Famelis et al. 2012), a model element with a *May* annotation is interpreted as a Boolean variable encoding whether the element is included in the model or not. These semantics are defined in propositional logic, as a formula over all such Boolean variables, called a *May formula*. The resolution of uncertainty, called *concretization*, thus takes the form of assigning a truth value to all the variables such that the May formula evaluates to True. *Abs*, *Var*, and *OW* partiality consider uncertainty in the uniqueness of an element, distinctness of an element, and completeness of the model, respectively.

3. Description of DRUIDE

In this section, we introduce the two components of DRUIDE: (a) a language for articulating design uncertainty about modeling decisions, and (b) a tracing mechaninism for describing the impact of uncertainty to system models. We illustrate them using the PTN metamodel example, shown in Fig. 3, where the modelers have elaborated the initial design from Fig. 1 while using DRUIDE to model their uncertainty and design decisions.

3.1. Language Definition

We created the DRUIDE modeling language by combining concepts from the modeling approaches described in Sect. 2. We show the metamodel in Fig. 2. In what follows, we refer to metaclasses of the DRUIDE metamodel simply as "classes".

To adapt the distinction between the problem space and the solution space from DoplerVML, in DRUIDE we distinguish between *DUncertainty* and *DDecision*. The "problem space" consists of statements of uncertainty and the "solution space" is a set of specific decisions, such that making a decision resolves the corresponding uncertainty. Thus, *DUncertainty* elements are the basic units for representing the uncertainty of a modeler about the design of a software artifact. *DDecision* elements are used to represent decisions about the design.

The *DUncertainty* class extends the U-Model by adding a *description*, used for recording a textual explanation of the modeler's uncertainty. In the example, the modeler creates the *DUncertainty* element DU1 with the description "How should we represent the PTN metamodel?". A *DUncertainty* element can be associated with a *DIndeterminacySource*. This class specializes the U-Model by adding a *description* attribute that can store an explanation of the cause of uncertainty. The element DU1 in Fig. 3 is associated with the indeterminacy source element DI1 whose description explains that the modeler is uncertain about the PTN metamodel design.

The class *DIndeterminacySource* has a *nature* attribute of type *DIndeterminacyNature*, which is an enumeration of the U-Model's indeterminacy natures listed in Sect. 2. The *nature* of the element DI1 in Fig. 3 is *MissingInformation* as the modelers are uncertain because they do not have enough information about what is the best way to represent arcs in PTNs. The *DIndeterminacySource* also includes a *level* attribute of type *DUncertaintyLevel*, which is an enumeration of the uncertainty levels proposed by Walker et al., as described in Sect. 2. The level of the element DI1 in Fig. 3 is *ScenarioUncertainty* as the modeler has at their disposal a set of alternative solutions from the Metamodel Zoo but does not know which one to use.

Next, the *DDecision* class is inspired from the DoplerVML language. It has a *question* attribute that textually represents the question that must be answered in order to resolve the decision, as well as a *resolved* attribute (Boolean) to indicate whether the decision has been made. In the example, after thinking about DU1 in concrete terms, the modelers identify three design decisions that would allow them to resolve DU1. They add three *DDecision* elements, marking each as not resolved: DD1, asking "How should Arcs be represented?"; DD2, asking "Should Arc meta-classes contain weight attributes?"; and DD3, asking "Should the metamodel enable the storage of the location of the graphical elements on the diagram?".

Each *DDecision* element is assigned a *DType* that captures the form of the question at the core of the decision. We distinguish between open and closed decision types. Closed ended decisions, represented by the class *DClosedEnded*, are those for which resolution means picking one from a set of enumerated

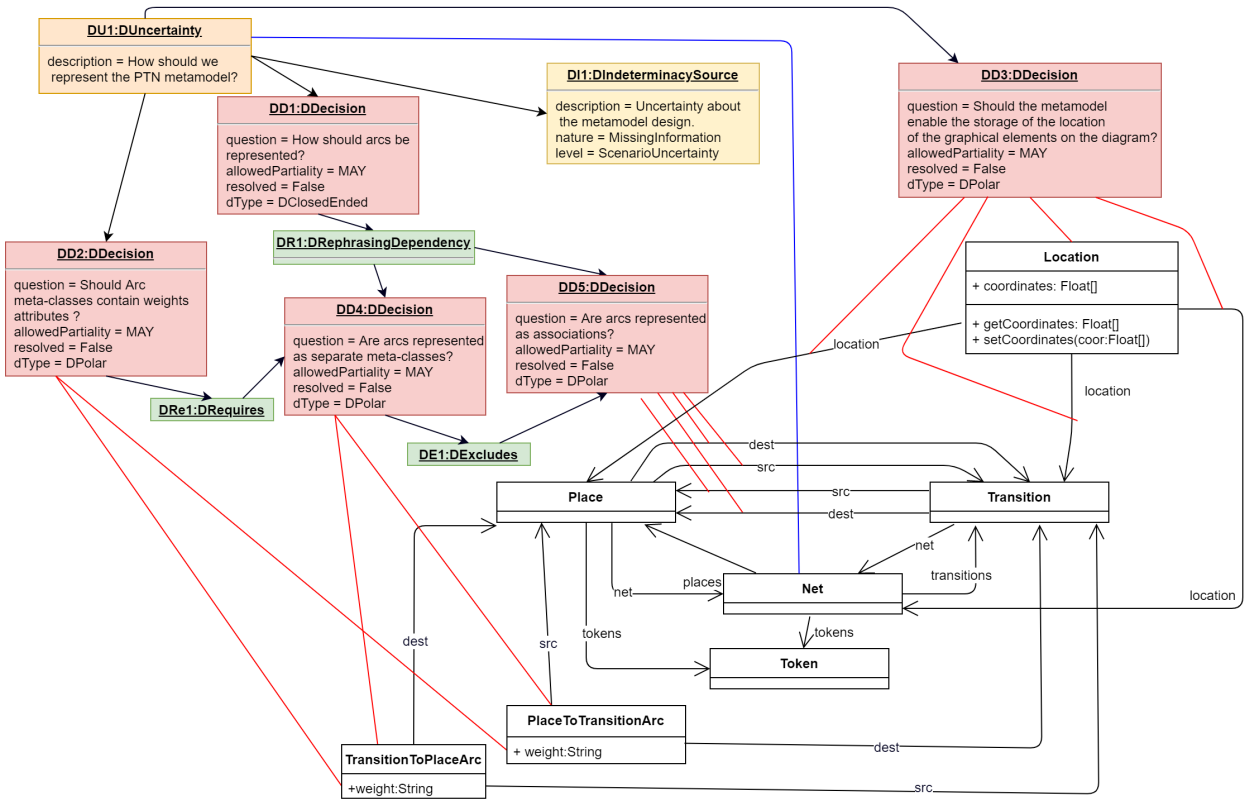


Figure 3 DRUIDE applied to the PTN metamodel.

alternative answers. *DPolar* is a special kind of closed decision where the answer is Boolean (yes/no). Open ended decisions, represented by the class *DOpenEnded*, are those for which the set of potentially acceptable answers has not yet been defined. The decisions DD2 and DD3 in our example are *DPolar*, as their corresponding questions can be resolved by answering yes or no. The decision DD1 is typed as *DClosedEnded*, as the modelers have at their disposal an enumerated set of alternative solutions for representing arcs from the Metamodel Zoo.

Each *DDecision* element also has an *allowedPartiality* attribute of type *DPartiality*, which is an enumeration of the four MAVO partiality types described in Sect. 2. This is an indication of how the modeler may realize the decision in the model. This is explained further in the next section, where we introduce operationalization traces. In our example, the modeler uses *May* partiality throughout, as it is well suited to representing an enumerated set of alternative designs (Famelis et al. 2012). In this paper, we limit ourselves to this partiality type, leaving the extension to other MAVO partiality types for the future.

Decisions can impact other decisions in various ways. In DRUIDE, this is captured by the *DDependency* class, which links *source* and *target DDecision* elements. We define and describe three subclasses: *DRephrasingDependency*, *DLogicalDependency*, and *DInformationRequirementDependency*.

We use *DRephrasingDependency* when one *DDecision* is a restatement of another *DDecision*. This can be used to capture the cases where one decision refines another, e.g., by providing more context, or by addressing a more specific concern. Ideally, this additional information is reflected in evolving the *DDecision*'s *DType*: from *DOpenEnded* to *DClosedEnded* or from *DClosedEnded* to *DPolar*. In the simplest case, a rephrasing may express the same decision from a different perspective without necessarily adding more information, keeping the *DType* the same. In our example, the modeler refines the *DClosedEnded DDecision* DD1 into two *DPolar* alternative *DDecision* elements: DD4, with the question "Are arcs represented as separate metaclasses?"; and DD5 asking "Are arcs represented as associations?". This evolution in the modeler's understanding is represented by the *DRephrasingDependency* element DR1.

In DRUIDE, *DLogicalDependency* objects describe dependencies between decisions that can be encoded in propositional logic. We assume that the source and target *DDecision* elements can be encoded as propositional variables, each one encoding the corresponding *question* of the decision. Dependencies are captured by a propositional formula stored in an associated *DDependencyFormula* element. We further elaborate two special cases of *DLogicalDependency*: *DRequires* and *DExcludes*. The former is a shorthand for $s \rightarrow t$, and the latter for $s \rightarrow \neg t$, where s and t are conjunctions of the source and target decision variables respectively. In our example, the *DPolar* decisions DD4 and DD5 are mutually exclusive. This is expressed as a *DLogicalDependency* with the *DExcludes* element DE1. Modelers find that if arcs are represented as metaclasses (DD4) then weights can be stored in arc metaclasses (DD2), and express this with the *DRequires* element DRE1 between DD2 and DD4.

Inspired by the DMN standard, the *DInformationRequire-*

Operation	Constraint	Rationale
Create a *LocalizationTrace*	All *DUncertainty* elements should be localized (e.g., should at least have one localization trace).	Avoid having extremely vague uncertainties. The user should, be able to denote which parts of the system the uncertainty relates to. In the worst case, the user can localize the uncertainty at the model level.
	A *DUncertainty* element cannot have two localization traces with the same target element (e.g., same source and same target).	Avoid redundant links.
Create an *OperationalizationTrace*	Only *DPolar* decisions can be operationalized.	Other types of decisions cannot be operationalized, they require further refinement for operationalization.
	A *DPolar* decision cannot have two operationalization traces with the same target element (e.g., same source and target).	Avoid redundant links.
	The system elements that have been introduced as a result of the operationalization of a *DPolar* decision should be linked to at least one of the system elements that the decision's *dUncertainty* was localized at.	The elements that operationalize a decision should be traced to the elements where that decision's uncertainty was localized. This is to make sure, to some extend, that the operationalization and the localization are consistent.
	A *DInformationRequirementDependency source* set of *DPolar* decisions cannot be operationalized, if one decision among the *DInformationRequirementDependency target* set is still unresolved.	A *DInformationRequirementDependency* means that the output of the target decisions set is used as input for the source decisions set; therefore, the source set cannot possibly be operationalized unless the target set has been already resolved.

Table 2 Constraints for *LocalizationTrace* and *OperationalizationTrace* elements.

mentDependency is used when the result of a decision influences another decision, and it is not conceptually correct or convenient to represent this influence using propositional logic. Examples of this dependency type are discussed in Sect. 4.

Finally, we define a set of six well-formedness and consistency constraints for *DDependency* links between *DDecision* elements. We list them with rationale in Tbl. 1. For example, the first constraint imposes that a *DDependency* element cannot *dependsOn* and *dependedUpon* the same *DDecision*, which avoids self-depended *DDecisions* elements in the model.

3.2. Tracing Design Uncertainty

Using the metamodel described in Fig. 2, modelers can express design uncertainty about any model, as the DRUIDE specification is self-contained and independent from other artifacts. To show how design uncertainty impacts a specific system, we connect the DRUIDE model with the model of the system via trace links. A trace is a simple object that references two other objects, one from the DRUIDE model and one from the system model. Other than establishing the link, we can give meaning to traces, using them to localize the expressions of uncertainty in the system and to describe the effect of design decisions on the elements of the system model. We distinguish two types: *LocalizationTrace* and *OperationalizationTrace*.

LocalizationTrace elements identify which parts of the system concern the uncertainty. A *LocalizationTrace* links a *DUncertainty* element from the DRUIDE model with the set of elements from the system model, which the modeler identifies as relevant to the *DUncertainty*'s description. If the modeler is unsure about choosing a specific system model element, the entire model may be chosen. In the PTN example, the PTN metamodel plays the role of the system model; its classes are shown in the bottom right part of Fig. 3. The elements of the DRUIDE model occupy the top left part of the figure. The modeler localizes the *DUncertainty* element DU1 at the NET class level using a *LocalizationTrace*, shown as a blue line. Note that *LocalizationTraces* do not have any effect on the system model, and serve only to document the modeler's thought process.

In DRUIDE, *OperationalizationTrace* elements are used to model the impact of design decisions on the system model. We borrow the "operationalization" concept from the field of Requirements Engineering, where leaf level goals of a goal model are operationalized into concrete tasks (van Lamsweerde 2009). An *OperationalizationTrace* points to the system model elements that would be impacted by a given *DDecision*.

The impact of a *DDecision* on system model elements is determined by its *DType* and its *allowedPartiality*. In the current iteration of DRUIDE, we only support the creation of *OperationalizationTrace* elements for fully refined decisions (i.e.,*DPolar*). We assume that such decisions only allow the *May* partiality type. More formally, an *OperationalizationTrace* linking a *DDecision* p to a set $\{q_1, \ldots, q_n\}$ of system model elements represents the formula $p \leftrightarrow q_1 \wedge \ldots \wedge q_n$. In other words the linked system elements of an *OperationalizationTrace* are included in the final system model if and only if the modeler decides to answer "yes" to the question of the corresponding *DDecision*. We can construct a partial model (Famelis et al. 2012) from the system model by annotating the linked system model elements of *OperationalizationTrace*s with *May* partiality and constructing the May formula from the *OperationalizationTrace* links and the *DLogicalDependency* relations between the corresponding *DDecision* elements in the DRUIDE model.

In the PTN example (see Fig. 3), the modelers operationalize the *DPolar* decisions DD4 and DD5 by introducing the PlaceToTransitionArc and TransitionToPlaceArc classes, and associating these classes with the Place and Transition classes via src and dest links. Additionally, they operationalize DD2 with the weight attribute in each of the PlaceToTransitionArc and TransitionToPlaceArc classes. Finally, they operationalize DD3 with the Location class and three location associations. The relevant

OperationalizationTrace elements are shown in red in Fig. 3. Resolving any of these decisions has a direct impact on the PTN metamodel. For example if the modeler answers "yes" to DD5 and "no" to DD3, the resulting PTN metamodel will be the same as the one shown in Fig. 1 with only the addition of the src and dest links between the Place and Transition classes.

Finally, we define a set of well-formedness constraints for the creation of *LocalizationTrace* and *OperationalizationTrace* elements. We show them along with their rationale in Tbl. 2. For example, the first constraint imposes that all *DUncertainty* elements should be localized. This ensures avoiding vague uncertainties that are unrelated to the system model.

3.3. Modeling Design Uncertainty

Given the DRUIDE metamodel, as well as the constraints in Tbl. 1 and Tbl. 2, we sketch a workflow for modeling design time uncertainty. The steps below correspond roughly to the steps taken by the modeler in the PTN example, described in the previous subsections.

Step 1 When first faced with uncertainty, document it in *DUncertainty* elements and optionally characterize their *DIndeterminacySource*.

Step 2 Localize each *DUncertainty* element in the system model using *LocalizationTrace* elements.

Step 3 Try to elicit specific design decisions from the description of the *DUncertainty* elements, and model them as *DDecision* elements.

Step 4 Describe the dependencies between decisions. Use *DInformationRequirementDependency* when the input of a *DDecision* depends on the output of others. Use *DLogicalDependency* associations to model any logical dependency between *DPolar* decisions.

Step 5 Initially, *DDecisions* may be of type *DOpenEnded*. Try to refine them to more specific *DClosedEnded* decisions, keeping track of *DRephrasingDependencies*. Ideally, all decisions should be refined until they are *DPolar*.

Step 6 If it is clear how to implement a *DPolar* decision, add the required elements to the system model and connect them with *OperationalizationTrace* elements.

Using the above workflow and PTN example (Fig. 3), we demonstrate how DRUIDE satisfies the modelers' criteria MC1–MC7 presented in Sect. 1. The modelers of the PTN were able to express that they were uncertain about how to design aspects of the model at different levels of abstraction (MC1). They started by modeling a vague *DUncertainty* DU1 (MC2) that they expanded afterwards into a set of concrete decisions (DD1–DD3) (MC4). They were then able to evolve their decisions (e.g., DD4 and DD5 evolved from DD1) (MC5), and express the dependencies between them (DR1, DRE1 and DE1) (MC6). Finally, they were able to express where their uncertainty is located using the blue *LocalizationTraces* (MC3) and how their specific *DPolar* decisions can be made operational in the model using the red *OperationalizationTraces* (MC7).

We also applied the DRUIDE workflow to two additional non-trivial realistic worked examples that have been used to validate prior relevant research work (Famelis & Chechik 2019). One example concerns uncertainty about the design of a state machine model of a simple peer-to-peer protocol. The other example is about uncertainty caused due to multiple alternative ways to repair a sequence diagram model of a diagramming tool. The experience gained from these examples (Dhaouadi 2020) gives us confidence to proceed with the evaluative case study presented in Sect. 4.

In this section, we focused on the process of articulation and modeling of design uncertainty. At this stage, the modeler is able to derive a fully fledged partial model, which can be used in a variety of development tasks such as verification, transformation, and refinement (Famelis & Chechik 2019).

4. Evaluative Case Study

In this section, we present an evaluative case study to explore the expressiveness of DRUIDE. In the next section, we use this case study to answer our research question introduced in Sect. 1.

Case Scenario: Emma's Social Distancing Dinner. In the midst of the COVID-19 pandemic (Zhu et al. 2020), Emma, a Quebec resident, must decide how to acquire dinner for the week. Emma has some options for dinner, but she wants to protect herself, as well as her local community. Emma wants to decide between cooking at home or picking up takeout from a local restaurant. She is concerned that she may not be able to practice social distancing when picking up takeout but also does not want businesses to suffer.

In order to help Emma make the optimal decision in this case study, we model the intentions of Emma in Tropos and model epidemiological data about the transmission of COVID-19 in Emma's community using a Bayesian network. We then connect these models, enabling Emma to connect her personal decisions with their social impacts. The models in this paper represent current information about the COVID-19 pandemic as of mid-May 2020 in Quebec, Canada.

Goal Modeling in Tropos. Goal models are used to elicit and document the intentions of stakeholders in the early phases of requirements engineering (Mylopoulos et al. 1992; Horkoff et al. 2019). In this paper, we use the Evolving Intentions framework (Grubb 2019) to evaluate Emma's intentions and consider tradeoffs in her decisions about dinner (i.e., whether to cook at home or order takeout, see Fig. 4). The Evolving Intentions framework allows for evolution in Tropos goal models (Giorgini et al. 2005), and the creation of simulation paths that return how the evaluation of each node in the model changes over time.

Bayesian Belief Networks. Bayesian networks are used in artificial intelligence for reasoning with uncertainty (Ertel 2017). These networks are acyclic directed graphs, where each node represents variables, and each edge represents causal influence. We build a Bayesian network based on an epidemiological modeling report about COVID-19, provided by the Government of Quebec (Brisson et al. 2020). The report captures the impact of respecting *lock-down measures* on the probabilities of exposure to the virus, and resulting hospitalizations and deaths.

Modeling Uncertainties. For both Emma's Tropos goal model and Bayesian networks of COVID-19 transmission risk, we use DRUIDE to articulate uncertainties. In each of the remaining subsections, we describe, in detail, the process of modeling

with DRUIDE and the resulting models. Sect. 4.1 details the uncertainty in Emma's goal model and Sect. 4.2 describes the Bayesian networks. We link these two models to create a full picture and describe the uncertainties in Sect. 4.3, which enables Emma to consider the potential impacts of her decisions on the Bayesian network variables. Finally, in Sect. 4.4, we discuss the use of DRUIDE in this case study and lessons learned.

4.1. Emma's Goal Model

Before discussing our uncertainties, we give an overview of Emma's goal model in Fig. 4. The original model (Varnum et al. 2020) is shown inside the dotted hexagon in Fig. 4 and consists of two actors: Emma and Society. Emma has two goals; have dinner, and not get or transmit COVID-19. The goal have dinner is decomposed into two potential tasks (green hexagons), pick up takeout or cook at home. The task cook at home helps (i.e., contributes positively to) the soft goal (orange peanut shape) of practice social distancing because it does not require Emma to leave her house. The task cook at home is also impacted by the availability of the fresh groceries resource (light blue rectangle), which decreases over time. The actor Society has three soft goals: minimize economic impact, minimize exposure to essential workers, and minimize the spread of COVID-19. Minimize exposure to essential workers helps minimize the spread of COVID-19. Emma's task pick up takeout helps to minimize economic impact by supporting a local restaurant, but hurts (i.e., contributes negatively to) minimize exposure to essential workers because it requires interaction with the restaurant staff. Finally, Emma's goal to not get or transmit COVID-19 helps Society to minimize the spread of COVID-19.

To ensure that the above model most accurately depicts Emma's scenario, several uncertainties must be articulated. We use DRUIDE to describe these modeling uncertainties. The resulting model is presented in Fig. 4.

First, we are unsure about the relationship between the task pick up takeout and the soft goal practice social distancing. We ask the question, "What is the relationship between 'pick up takeout' and 'practice social distancing'?". We model it as the *DUncertainty* E-DU1. Because this uncertainty derives from lacking precise information of the nodes' relationship, we characterize the *DIndeterminacySource* E-DI1 nature as *InsufficientResolution*. This is a *ScenarioUncertainty*, because there is a fixed set of possible relationships in goal modeling. We localize E-DU1 using the blue *LocalizationTraces* as shown in Fig. 4. To resolve this uncertainty, we suggest decomposing pick up takeout into two tasks: pick up takeout with contact and pick up takeout with no contact. We model this as the *DPolar* decision E-DD1. However, if we decide to decompose E-DD1, we then must decide what contribution links to add, expressed in E-DD2. Since E-DD2 depends on the output of E-DD1, we use the *DInformationRequirementDependency* to express this.

Second, we consider if there exists a relationship between the soft goal practice social distancing and the goal not get or transmit COVID-19. Specifically, we are unsure if this preventive measure helps reduce the spread of the virus, has no effect on virus spread, or possibly makes matters worse. The decision is impeded from the absence of information. We express this using the *DUncertainty* E-DU2 and characterize the *DIndeterminacySource* E-DI2 by specifying its level to *RecognizedIgnorance* and its nature to *MissingInformation*. Then, we localize it using the blue *LocalizationTraces*. We articulate our decisions. Initially, we need to decide if a relationship exists (E-DD3). We operationalize this decision by the introduction of a link between the practice social distancing soft goal and the not get or transmit COVID-19 goal. This link is connected to E-DD3 by an *OperationalizationTrace* shown in red (see Fig. 4). We also need to decide which contribution link to use on this relationship, so we model the *DClosedEnded* decision E-DD4. Similarly, since we didn't make our decision about the existence of the link yet, and since E-DD4 depends on the output of E-DD3, we use a *DInformationRequirementDependency* to express this.

Third, we are uncertain about the behavior of the resource fresh groceries. Within the Evolving Intentions framework, the MN label on the fresh groceries resource in Fig. 4 indicates that fresh groceries will become less fulfilled over time. However, we are unsure if this is the best evolving function to characterize fresh groceries (E-DU3). Since this *DIndeterminacySource* is of a composite *nature* (E-DI3), we articulate three possible decisions (E-DD5, E-DD6 and E-DD7) and express the dependencies between them (E-DR3 and E-DR4).

Forth, we ask the question (E-DU4), "Should we decouple get and transmit COVID-19?". We consider decomposing the goal not get or transmit COVID-19 into two goals: not get COVID-19 and not transmit COVID-19 (E-DD8). We are unsure that the factors that may prevent a person from being infected are the same as the factors that prevent a person from infecting others (E-DI4). Additionally, not transmitting the virus may contribute differently in reducing the spread of the virus. Intuitively, we suggest that it may contribute more strongly, but there is no official information yet that clarifies this difference.

Fifth, we are uncertain about how to model the impact of wearing a mask (E-DU5), and how to represent this *ScenarioUncertainty* (E-DI5) in our goal model. We localize this uncertainty at different parts of the model, as shown by the multiple blue *LocalizationTraces* connected to E-DU5 (see Fig. 4). Initially, we consider adding an intention for wearing masks (E-DD9). Then, if we choose to add an intention, we must decide on its type (E-DD10). Once resolved, we need to decide the relationships that should be connected to this intention to represent the impacts of wearing a mask (E-DD11). To express this chain of dependencies, we use the *DInformationRequirementDependency*.

Finally, we are unsure about how to evaluate the soft goal minimize the spread of COVID-19. Thus, we elaborate our uncertainty (E-DU6) and articulate our decision concerning this (E-DD12).

4.2. Quebec's Epidemiological Model

We model a Bayesian Network which captures the impact of respecting the lock-down measures on the probabilities of exposure to the virus, hospitalization and death, based on a government report (Brisson et al. 2020). We first modeled the Bayesian Network presented inside a dotted line in Fig. 5, which shows five Boolean variables that represent the possible health states

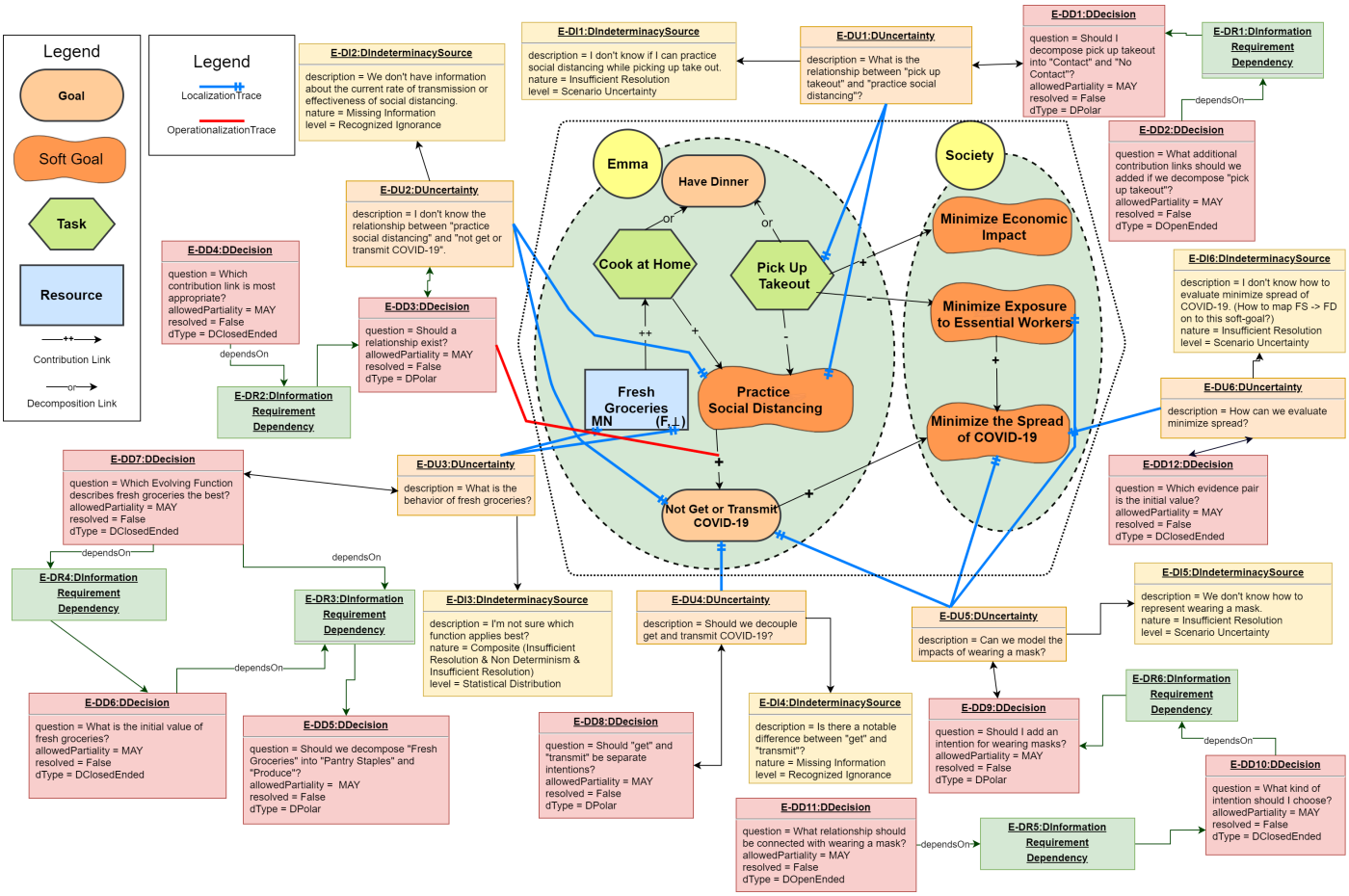


Figure 4 The resulting model after expressing the modeling uncertainties with DRUIDE on Emma's goal model.

of an individual during the pandemic: Susceptible, Exposed. Symptomatic, Hospitalized and Dead. The transition from one state to another is estimated based on conditional probabilities shown in matrices next to the edges in the figure. We deduced these probabilities from the report (Brisson et al. 2020). For instance, the probability of a person to be hospitalized if they are known to be symptomatic is equal to 0.07. The report also specifies the Symptomatic probabilities based on the age of a person. Thus, we add the age variable and the corresponding conditional probabilities in our model.

We use DRUIDE to express our uncertainties and present them in Fig. 5. First, the report does not specify the probability of being exposed to the virus. We model this as a *DUncertainty* (DU1) with an indeterminacy source (DI1) of type MissingInformation. Then, we localize it at the Exposed-Susceptible conditional probabilities matrix.

Second, to map the epidemiological model to the Bayesian Network semantics, we gathered the mutually exclusive health states as the possible values for one variable. For instance, in (Brisson et al. 2020), researchers distinguish between Symptomatic and Asymptomatic. In our Bayesian model, we used only the Symptomatic variable with two values: True and False. When the value is False, the probabilities express the state Asymptomatic. Similarly, the epidemiological model has two states: Death and Recovered. In our Bayesian model, we use only the Death variable with True and False values, where the False value refers to the Recovered state. Since we do not know what the Bayesian model should emphasize, we are unsure which variables are more appropriate (i.e., Death vs. Recovered and Symptomatic vs. Asymptomatic). We model this uncertainty as the *DUncertainty* DU2. Since we have two alternatives for the indeterminacy source of each variable, we denote the level as *ScenarioUncertainty* and nature as *InsufficientResolution* in DI2. We localize *DU2* in the Symptomatic and Death variables with the blue lines connected to *DU2* in Fig. 5. Additionally, we express two concrete *DClosedEnded* design decisions: DD1 and DD2. Since, the articulation of DD1 is similar to DD2, we only expand on DD2 to reduce visual clutter. Specifically, we refine DD2 into two mutually exclusive *DPolar* decisions (DD3 and DD4) and operationalize each. For example, the operationalization of DD3 resulted in the addition of the Recovered variable, the edge linking the Hospitalization state to the Recovered state and the Recovered-Hospitalization matrix. These elements are connected to DD3 using the *OperationalizationTrace* red links.

Third, we are unsure how to model the fact of respecting preventive measures. The report distinguishes between two measures: Reduce contacts and Isolation if symptomatic. We model the uncertainty of whether they should be distinguished as two variables (DU3), and characterize this indeterminacy source (DI3). We articulate and evolve the related decisions

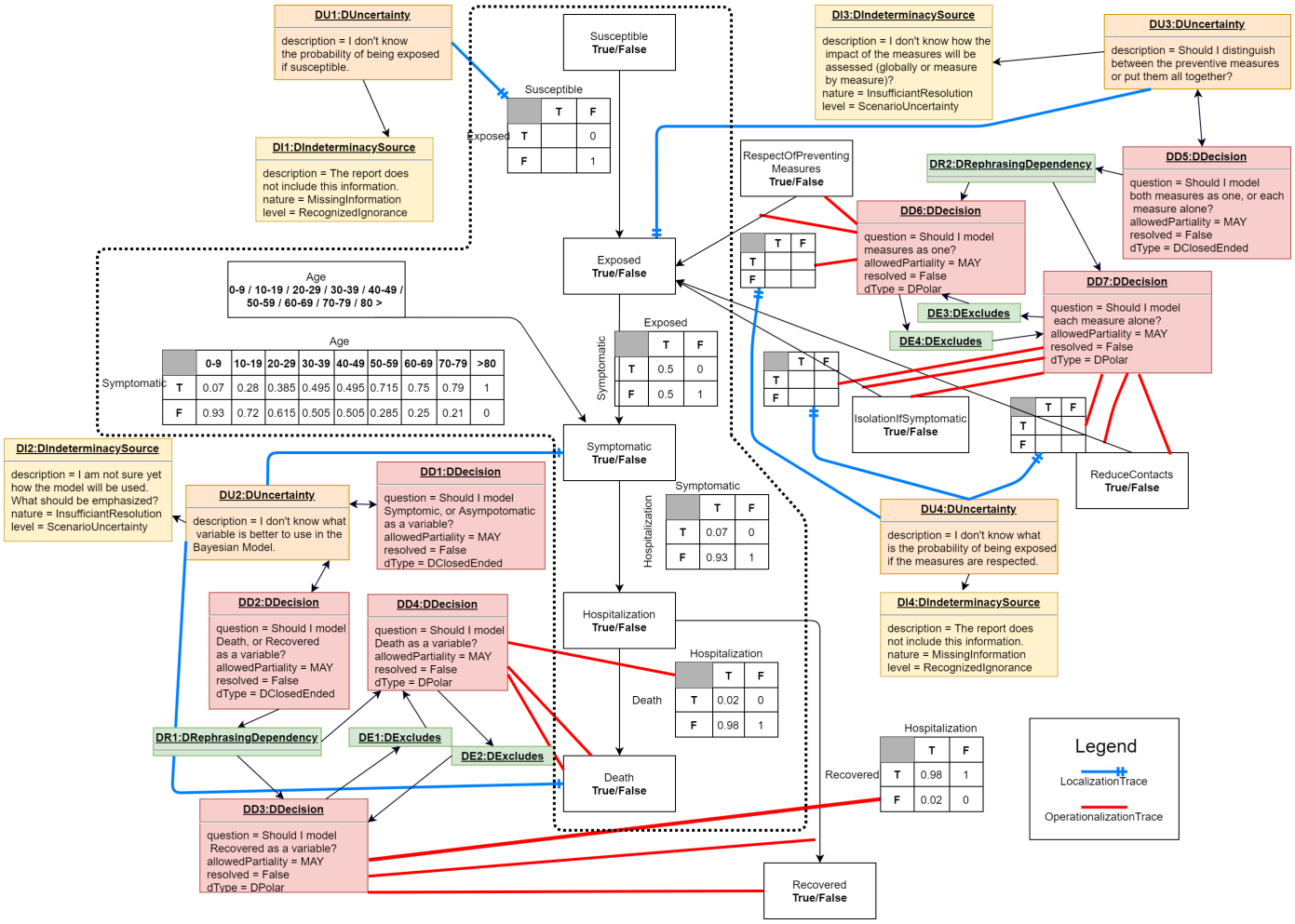


Figure 5 DRUIDE on the Bayesian network of Quebec's COVID-19 epidemiological model.

(DD5–DD7), operationlize the *DPolar* ones (DD6 and DD7).

Finally, the report didn't include the impact of respecting the preventive measures on being exposed to the virus; thus, we express and localize this uncertainty (DU4).

4.3. Linking Models

After creating Emma's goal model and Quebec's epidemiological model separately, we link the case study components together forming the complete picture. We map the goal model tasks to the corresponding evaluation of the Bayesian variables they may impact. For example, when Emma chooses to cook at home, she respects the preventive measures by reducing contact; thus, cook at home is connected with the True evaluation of the Bayesian variables Reduce Contacts and Respect of Preventing Measures. By considering the interactions between Emma's decisions and the Bayesian network variables, Emma can choose a more optimal solution. We denote this mapping in Fig. 6 using green dashed links. These links indicate the presence of a mechanism that observes the goal model and triggers an update of the corresponding Bayesian network algorithm, once a task has been chosen. We also note that we omit several DRUIDE elements to reduce visual clutter in Fig. 6.

We are faced with uncertainty when mapping pick up takeout with variables in the Bayesian network. We ask two questions of this relationship, "Does pick up takeout break preventive measures, or does it respect them?" and "Does it help reduce contact, or not?". Intuitively, it is safer than eating outside at a restaurant but riskier than cooking at home. We are uncertain if it should be considered as a preventive or non-preventive measure. We use DRUIDE to represent this uncertainty (DI5, DU5) and articulate its corresponding decisions (DD8, DD9, DD10). We operationalize each alternative as follows. DD10 considers pick up takeout as a non-preventive measure, and thus maps it to the False values of the variables Reduce Contacts and Respect of Preventing Measures (see orange dotted links in Fig. 6). On the other hand, DD9 views pick up takeout as a preventative measure, and maps it to the True values of Reduce Contacts and Respect of Preventing Measures (see green dashed links in Fig. 6). In the latter case, pick up takeout leads to an identical evaluation as cook at home. This means that enacting DD9 would cause the Bayesian network to view both tasks as equivalent. The probability of exposure would be the same, and both tasks would carry the same risk.

When merging the two models into one, we found similar uncertainties between them. For example, the E-DU2 uncertainty that describes the relationship between practicing social

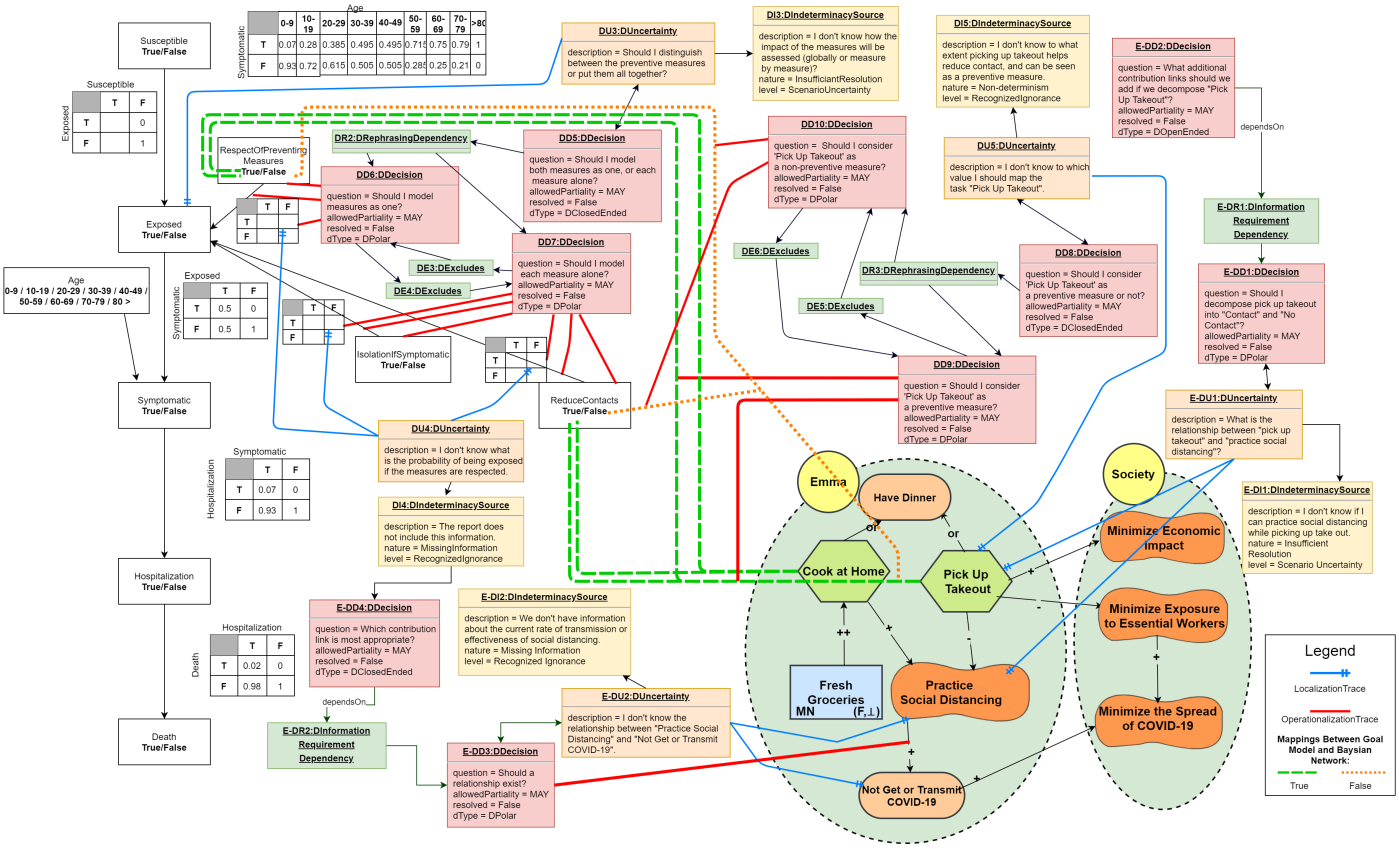


Figure 6 DRUIDE on the mapping between Emma's goal model and Quebec's epidemiological model.

distancing and not getting or transmitting the virus in the Emma model is the same as the DU4 uncertainty that describes the impact of respecting preventive measures on the probability of exposure to the virus in the Bayesian network. Although phrased differently, they both refer to the effectiveness of preventive measures on reducing the spread of the virus. Similarly, the E-DU1 uncertainty that describes the relationship between picking up takeout and practicing social distancing in Emma's goal model refers to the DU5 mapping uncertainty explained above. This illustrates how the same uncertainty can appear in different models.

Finally, we note that our final model in Fig. 6 depicts three different types of uncertainties: (1) Emma's uncertainty in her dinner decision, captured by the goal model; (2) the uncertainty in Quebec's COVID-19 epidemiological model, expressed using the Bayesian network probabilities; and (3) the uncertainty surrounding the models and their relationships, articulated using DRUIDE.

4.4. Lessons Learned

Using DRUIDE, we were able to represent the inherent uncertainties in Emma's goal model and transform them into concrete decisions. In exploring alternatives, we found that for the most part, we made appropriate decisions about how to represent Emma's tradeoffs in Tropos, and the epidemiologic information in our Bayesian network. By specifying and maintaining this uncertainty, we were able to use Emma's goal model to make decisions, as well as update it as new information became available. For example, in our initial analysis with Emma's goal model, we found that Emma should cook at home to keep her and others the most safe (Varnum et al. 2020). We were able to update our representation of the risks associated with Pick Up Takeout in the Bayesian network with the availability of new surface transmission information (not shown). We resolved E-DU1 by making decisions for E-DD1 and E-DD2 (see Fig. 6). In doing so, we decomposed Pick Up Takeout into Contact Pick Up and No Contact Pick Up and linked these new tasks to the soft-goals in the model (not shown for space considerations). By articulating and resolving our uncertainties in Emma's goal model, we are more confident about our original model and our ability to make the most appropriate decision.

The linked model in Fig. 6 illustrates how DRUIDE satisfies the needs of modelers (see MC1–MC7 in Sect. 1), across different model types (i.e., inter-model). We were able to express our uncertainty (MC1) by elaborating the vague *DUncertainty* DU5 (MC2), and were able to localize this uncertainty using the blue *LocalizationTraces* (MC3). We articulated a specific design decision (DD8) (MC4) and evolved it into two *DPolar* decisions (DD9 and DD10) (MC5). Additionally, we could express the dependencies between the decisions (DR3, DE5 and DE6) (MC6). Finally, the red *OperationalizationTraces* enabled us to show how the *DPolar* decisions can be made operational in the model (MC7).

5. Results

To answer our research question *"To what extent can* DRUIDE *be used to articulate uncertainty?"*, we consider uncertainty within the same model (intra-model) or across related models (inter-model) in real scenarios. We draw on our experiences from the case study in Sect. 4 and the examples described in Sect. 3.3.

Intra-model Validation. In this paper, we applied DRUIDE to two scenarios, in two different software lifecycle phases. One concerns a metamodel for PTNs, during the design phase; the other concerns a Bayesian network and a Tropos goal model, during the domain analysis phase of requirements engineering.

The workflow described in Sect. 3.3 was developed based on our experiences with the case study presented in Sect. 4. For each model, we used brainstorming to probe our understanding of the models and created a list of all our uncertainties. We then explored each uncertainty in detail and modeled it using the elements of the DRUIDE metamodel. We found it helpful to consider the *IndeterminacySource* first, followed by generating possible *DDecisions* for each uncertainty. Finally, we considered any localization and operationalization in the model. We were able to represent all the uncertainties that we listed using the metamodel presented in Fig. 2.

An earlier draft of our metamodel did not include *DInformationRequirementDependency*. In creating Emma's goal model (see Fig. 4), we incorrectly used the *DLogicalDependency* to describe when a decision depended on the resulting information from another decision. By adding the *DInformationRequirementDependency* we were able to more accurately represent these dependencies (e.g., E-DR2 and E-DR3, in Fig. 4).

Within the context of creating goal models, we found DRUIDE complementary to prior work in goal modeling, which explored beliefs of the modelers and assumptions in the environment (Amyot et al. 2010; van Lamsweerde 2009), and we were able to articulate and document uncertainty about elements in the model.

Inter-model Validation. We focus on the application of DRUIDE on the linked model from our case study described in Sect. 4.3 (see model in Fig. 6). This model depicts and links the uncertainty both in the Bayesian network and the goal model. Using DRUIDE, we were able to articulate uncertainties concerning the relationships between two different and heterogeneous models (see DU5 and its corresponding decisions). Also, the same uncertainty can appear in different models. We already noted similar uncertainties above (e.g., E-DU2 and DU4). In DRUIDE, these can be easily merged into one, and have associated decisions in different models.

Finally, the case study clearly illustrated that different types of uncertainty require different modeling approaches and different treatments (see Sect. 4.3). Existing modeling notations, like Bayesian networks or goal models, are suitable for representing uncertainty inside the system, however they do not cover the uncertainty about the design of the system. DRUIDE decouples the modeling and representation of design uncertainty from problem specific representations of uncertainty. This is a form of "separation of modeling concerns" that helps clarify the role of different types of uncertainty in models.

Initial Applicability. Without an empirical investigation with independent users, we cannot make claims about the applicability of DRUIDE. Here we give our initial impressions, based on our experiences with the evaluative case study presented in Sect. 4. The first idea that DRUIDE users have to internalize is that it explicitly models uncertainty *about* a model rather than uncertainty *within* the model. This distinction is especially important for languages that model scenario uncertainty or tradeoffs (e.g., goal models). Second, we found that the open-ended nature of the *DUncertainty* element helps users quickly add elements to the model without any concern for how they will be formalized. However, once users begin creating and linking *DDecision* elements, it is important that they review the DRUIDE metamodel (see Fig. 2) and the list of constraints (see Tbl. 1 and Tbl. 2). Finally, novice users may also struggle with identifying *DIndeterminacyNature* and *allowedPartiality*. Although definitions are given as part of DRUIDE, providing additional examples for each of these concepts to users can help guide them to the appropriate categorization.

Based on these three perspectives, we answer our RQ:

> We found DRUIDE to be sufficient for articulating uncertainty both within a model and across models of different types.

6. Threats to Validity

We briefly explore possible threats to our evaluative case study (Runeson et al. 2012), and begin by looking at two potential issues with *Construct Validity*. First, we cannot guarantee that the concepts, models and relationships used in the case study were interpreted correctly. To mitigate this, we held regular meetings to clarify constructs. This iterative process lead to both an increased shared understanding of DRUIDE concepts and the case study contents within our team, as well as updates to the metamodel, as described in Sect. 5.

Second, there is a risk that the application of DRUIDE to the case study was not a reliable representation of a modeling process containing design uncertainty. To mitigate this threat, we chose a real world scenario outside our immediate areas of expertise. The case study concerned the ongoing public health crisis and was conducted at a time of heightened uncertainty both for health experts as well as for the general public who are called upon to make critical everyday decisions based on popularized science and public health guidelines. The uncertainty expressed in modeling the "Emma" persona and the Quebec epidemiological report is an accurate reflection of the real uncertainties faced by reasonably educated and socially responsible citizens during the COVID-19 pandemic.

Our finding that DRUIDE is sufficient for articulating uncertainty is threatened by both External Validity (generalizability to other domains and modeling notations) and Reliability (whether the analysis is dependent on the individual researchers involved). Both of these risks can only be mitigated by the independent findings of other researchers and future work to validate whether our results generalize to other modeling notations and scenarios.

7. Related Work

Design uncertainty (Ramirez et al. 2012) concerns the question about how to design a model and can result from dealing with different design alternatives (van Lamsweerde 2009), making decisions about architecture (Esfahani et al. 2013), resolving inconsistencies (Egyed et al. 2008), or conflicting stakeholder requirements (Sabetzadeh et al. 2010). Various approaches have been proposed for working in the presence of design uncertainty such as reasoning (Famelis et al. 2012), bidirectional transformations using JTL (Eramo et al. 2015), architectural interfaces (Watanabe et al. 2017), and pattern matching (Semeráth & Varró 2017). However, to the best of our knowledge, existing work does not explicitly discuss the process of articulating design uncertainty or representing its evolution.

DRUIDE is an extension of our previous work on modeling and managing the lifecycle of design uncertainty with partial models (Famelis & Chechik 2019). The central idea is to represent design uncertainty directly in models using formal partiality annotations (Salay et al. 2012). While in previous work we showed how partial models can be useful for accomplishing various development tasks, we did not address questions such as: "Why does this model element have a partiality annotation?", and "How did we come to assign a partiality annotation to that element?". DRUIDE fills the gap in the literature by providing a language and an approach to explicitly articulate and evolve design uncertainty in models.

The work by Zhang et al. on uncertainty modeling for Cyber-Physical Systems (CPSs) is closest to ours (Zhang et al. 2016). They propose the U-Model, a conceptual model for uncertainty, but unlike DRUIDE, they separate it from the CPS system model. In subsequent work, the authors propose the U-RUCM methodology and tool to identify and specify uncertainty as part of system requirements (Zhang et al. 2018). However, their work is focused on use-case modeling. In contrast, DRUIDE is a generic approach that applies to any modeling language.

As discussed in Sect. 2, our research is related to and inspired from work in decision modeling (OMG 2020-03) and decision oriented Product Line Engineering (PLE) (Dhungana et al. 2007). More broadly, since design uncertainty often involves modeling sets of design alternatives, DRUIDE is related to research on creating representations of sets of related models. This is a fundamental modeling task that has applications in the formal specification of requirements (Larsen 1989), the definition of feature-oriented PLE (Kang et al. 2002), the modeling of model families (Alwidian & Amyot 2020), etc.

Finally, there are some commonalities with work on capturing design rationale (DR) (Burge 2005), especially when it comes to documenting design decisions. One significant difference is that we focus on design uncertainty first, before it is crystallized to specific decisions. This allows modelling the earlier stages of the design lifecycle, as well as the formation and evolution of design decisions. One of the biggest adoption barriers to DR capture has been that the documentation of design decisions is often seen as an expensive afterthought (Burge 2005). We envision using DRUIDE to guide users in the design process in order to generate and elaborate design decisions, rather than retroactively documenting them. This would create a useful synergy with DR capture, potentially lowering its adoption barriers.

8. Conclusions and Future Work

In this paper, we introduced DRUIDE, a language and workflow for articulating design time uncertainty, and illustrated its usage on a software engineering example. We completed an evaluative case study of DRUIDE, which modeled an individual's decisions during the COVID-19 pandemic. Our evaluation showed that DRUIDE is sufficiently adequate and expressive to articulate design time uncertainty.

In the future, we intend to evolve the DRUIDE language, using the experience gained from modeling the COVID-19 case study and other examples. For example, we suspect that the decision types we adopt in this paper (i.e., *DOpenEnded*, *DClosedEnded*, and *DPolar*) may be overly simplistic. We plan to investigate other approaches, and take into account the potential impact of a decision type on the corresponding uncertainty level and nature. We used only *May* partiality (see Sect. 3.1), but we see potential for using *Abs* partiality in the future. Future work will expand the repertoire of *allowedPartiality* types and, crucially, define the semantics of DRUIDE models, using the MAVO logical formalism. Additionally, we want to further investigate other types of uncertainty, and elaborate the typology of uncertainties. Once complete, we may study the extent to which the existing modeling language adequately supports different uncertainty types.

DRUIDE gives modelers the ability to explicitly articulate uncertainty and compose completely heterogeneous models, by linking uncertainties. This launches new opportunities in automated data-driven software engineering decision making with both qualitative and quantitative information, where stakeholders can test hypotheses and resolve uncertainty to make informed decisions. In our case study, we used a single data model (a Bayesian network) for virus transmission. We envision that users could plug in multiple different domain models based on publicly available learned data. We also want to investigate the potential for creating intelligent assistants to help populate DRUIDE models with mined data from social interactions, such as online discussions about design (Viviani et al. 2019).

At the time of our case study, we used draw.io[1] to complete modeling tasks. We intend to develop tool support for DRUIDE, including implementing the metamodel, operators, and constraints introduced in this paper, as well as allowing for multiple data models for hypothesis testing and automated analysis.

Acknowledgments

This research was partially funded by the Tunisian Ministry of Higher Education and Scientific Research, a MITACS Research Training Award, and the SURF Program at Smith College.

[1] https://github.com/jgraph/drawio

References

Alwidian, S., & Amyot, D. (2020). "Union is Power": Analyzing Families of Goal Models Using Union Models. In *Proc. of MODELS'20* (p. 252-262).

Amyot, D., Ghanavati, S., Horkoff, J., Mussbacher, G., Peyton, L., & Yu, E. (2010). Evaluating Goal Models Within the Goal-Oriented Requirement Language. *International Journal of Intelligent Systems*, *25*(8), 841–877.

Brisson, M., Gingras, G., Drolet, M., Laprise, J., & et al. (2020). *Épidémiologie et Modélisation de l'évolution de la COVID-19 au Québec* (Tech. Rep.). Centre de Recherche du CHU de Québec – Université Laval. Retrieved from http://www.marc-brisson.net/covid19-response/Epidemiologie-et-modelisation-evolution-COVID-19-au-Quebec_7-mai.pdf (Pg. 29–33, accessed 09/13/2020.)

Burge, J. E. (2005). *Software engineering using design rationale.* PhD thesis, Worcester Polytechnic Institute.

Dhaouadi, M. (2020). *Articulating Design-time Uncertainty with DRUIDE.* MSc thesis, Université de Montréal.

Dhungana, D., Grünbacher, P., & Rabiser, R. (2007). Domain-specific adaptations of product line variability modeling. In *Working conference on method engineering* (pp. 238–251).

Dhungana, D., Grünbacher, P., & Rabiser, R. (2011). The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering*, *18*(1), 77–114.

Egyed, A., Letier, E., & Finkelstein, A. (2008). Generating and Evaluating Choices for Fixing Inconsistencies in UML Design Models. In *Proc. of ASE'08* (pp. 99–108).

Eramo, R., Pierantonio, A., & Rosa, G. (2015). Managing uncertainty in bidirectional model transformations. In *Proc. of SLE'15* (pp. 49–58).

Ertel, W. (2017). Reasoning with Uncertainty. In *Introduction to artificial intelligence* (pp. 125–174). Springer.

Esfahani, N., Malek, S., & Razavi, K. (2013). GuideArch: Guiding the Exploration of Architectural Solution Space Under Uncertainty. In *Proc. of ICSE'13* (pp. 43–52).

Famelis, M., & Chechik, M. (2019). Managing design-time uncertainty. *SOSYM*, *18*(2), 1249–1284.

Famelis, M., Salay, R., & Chechik, M. (2012). Partial models: Towards modeling and reasoning with uncertainty. In *Proc. of ICSE'12* (pp. 573–583).

Giorgini, P., Mylopoulos, J., & Sebastiani, R. (2005). Goal-oriented Requirements Analysis and Reasoning in the Tropos Methodology. *Engineering Applications of Artificial Intelligence*, *18*(2), 159–171.

Grubb, A. M. (2019). *Evolving Intentions: Support for Modeling and Reasoning about Requirements that Change over Time* (Doctoral dissertation, University of Toronto). http://hdl.handle.net/1807/95842.

Horkoff, J., Aydemir, F. B., Cardoso, E., Li, T., Maté, A., Paja, E., … Giorgini, P. (2019). Goal-oriented requirements engineering: an extended systematic mapping study. *Requirements Engineering*, *24*, 133–160.

Kang, K. C., Lee, J., & Donohoe, P. (2002). Feature-oriented product line engineering. *IEEE software*, *19*(4), 58–65.

Larsen, K. G. (1989). Modal specifications. In *Proc. of CAV'89* (pp. 232–246).

Mylopoulos, J., Chung, L., & Nixon, B. (1992, June). Representing and using nonfunctional requirements: a process-oriented approach. *IEEE TSE*, *18*(6), 483–497.

OMG. (2020-03). *Decision Model and Notation, version 1.3.*

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: an agile toolkit.* Addison-Wesley.

Ramirez, A., Jensen, A., & Cheng, B. (2012). A Taxonomy of Uncertainty for Dynamically Adaptive Systems. In *Proc. of SEAMS'12* (pp. 99–108).

Runeson, P., Höst, M., Rainer, A., & Regnell, B. (2012). *Case Study Research in Software Engineering - Guidelines and Examples*. Wiley.

Sabetzadeh, M., Nejati, S., Chechik, M., & Easterbrook, S. (2010). Reasoning about Consistency in Model Merging. In *Proc. of LWI'10.*

Salay, R., Famelis, M., & Chechik, M. (2012). Language independent refinement using partial modeling. In *Proc. of FASE'12* (pp. 224–239).

Schmid, K., Rabiser, R., & Grünbacher, P. (2011). A comparison of decision modeling approaches in product lines. In *Proc. of VAMOS'11* (pp. 119–126).

Semeráth, O., & Varró, D. (2017). Graph constraint evaluation over partial models by constraint rewriting. In *Proc. of ICMT'17* (pp. 138–154).

van Lamsweerde, A. (2009). *Requirements Engineering - From System Goals to UML Models to Software Specifications*.

Varnum, M. H., Spencer, K. M. B., & Grubb, A. M. (2020). Towards an Evaluation Visualization with Color. In *Proc. of i*'20 (istar'20)* (pp. 79–84).

Viviani, G., Famelis, M., Xia, X., Janik-Jones, C., & Murphy, G. C. (2019). Locating latent design information in developer discussions: A study on pull requests. *IEEE TSE*.

Walker, W. E., Harremoës, P., Rotmans, J., Van Der Sluijs, J. P., Van Asselt, M. B., Janssen, P., & Krayer von Krauss, M. P. (2003). Defining uncertainty: a conceptual basis for uncertainty management in model-based decision support. *Integrated assessment*, *4*(1), 5–17.

Watanabe, K., Ubayashi, N., Fukamachi, T., Nakamura, S., Muraoka, H., & Kamei, Y. (2017). iArch-U: interface-centric integrated uncertainty-aware development environment. In *Proc. of MiSE'17* (pp. 40–46).

Yue, T., Briand, L. C., & Labiche, Y. (2013). Facilitating the transition from use case models to analysis models: Approach and experiments. *ACM TOSEM*, *22*(1), 5.

Zhang, M., Selic, B., Ali, S., Yue, T., Okariz, O., & Norgren, R. (2016). Understanding uncertainty in cyber-physical systems: a conceptual model. In *Proc. of ECMFA'16* (pp. 247–264).

Zhang, M., Yue, T., Ali, S., Selic, B., Okariz, O., Norgre, R., & Intxausti, K. (2018). Specifying uncertainty in use case models. *Journal of Systems and Software*, *144*, 573–603.

Zhu, N., Zhang, D., Wang, W., Li, X., Yang, B., Song, J., … Tan, W. (2020). A novel coronavirus from patients with pneumonia in china, 2019. *N Engl J Med*, *382*(8), 727-733.