PR-OWL Decision: Toward Reusable Ontology Language for Decision Making under Uncertainty

Shou Matsumoto, Kathryn B. Laskey, Paulo C. G. Costa
Department of Systems Engineering and Operations Research
George Mason University
Fairfax, VA
[smatsum2, klaskey, pcosta]@gmu.edu

***Abstract*—Decision making is a big topic in Intelligence, Defense, and Security fields. However, very little work can be found in the literature about ontology languages that simultaneously support decision making under uncertainty, abstractions/generalizations with first-order expressiveness, and forward/backward compatibility with OWL—a standard language for ontologies. This work proposes PR-OWL Decision, a language which extends PR-OWL—an extension of OWL to support uncertainty—to support first-order expressiveness, decision making under uncertainty, and backward/forward compatibility with OWL and PR-OWL.**

Keywords—ontology, decision making, uncertainty, OWL

I. INTRODUCTION

Ontologies are engineering artifacts which consist of formal vocabularies of terms, usually describing specific domain knowledge and accessed by persons or computers sharing a common view or domain application. Various interdisciplinary works addressing the engineering aspects of this field have been held in the recent years by the information systems—in a broader sense—community [1, 2, 3, 4, 5]. The Web Ontology Language (OWL) is a standard ontology language which represents classes, properties, and individuals in Semantic Web documents [6]. In 2005, Probabilistic Web Ontology Language (PR-OWL) [7] was formulated to address OWL's lack of support for uncertainty—a ubiquitous factor in complex real-world problems. As a continuing effort, version 2 of PR-OWL [8] was formulated in order to address some backward compatibility issues with its predecessor OWL.

Nevertheless, continuous efforts have been performed in the field of decision support, especially with models supporting uncertainty [9, 10, 11, 12, 13, 14, 15]. Decision making is the process of selecting a course of action among several possibilities, based on values or preferences of some decision maker. Values and preferences play a very important role here, because they represent the desirability of an outcome, in a manner that is different from the likelihood or probability that the outcome will happen.

For example, one's probabilistic model may state that the probability of failing some exam is 20% if you do not study. The decision maker may consider this is an acceptable probability for choosing not to study, given that the impact of failing is nothing more than minor embarrassment. However, if the decision maker may lose his/her job as a consequence of failing the exam, the decision maker would definitely study hard. This well illustrates how difficult it would be for someone to make decisions based *only* on metrics of uncertainty (*e.g.* probabilities or likelihoods of events), and how important values and preferences are in actually taking some action. Consequently, ontologies for decision making need to support both uncertainty and values (or preferences of decision makers). Unfortunately, current ontology tools and languages often do not have standardized constructs for representing preferences.

On the other hand, there are models that were not originally designed for ontologies, but can be used for decision making under uncertainty with explicit representation of values. For instance, classic probabilistic decision models like Influence Diagrams (ID) [16] can be enough to just represent and solve decision-making problems—with representation of actions and values or preferences of a decision maker—with support for uncertainty. However, IDs perform probabilistic reasoning about propositional (as in propositional logic) statements, which is not expressive enough to capture many important situations; thus we would like to have first-order expressiveness (as in First-Order Logic), with functions, predicates, and quantification.

OWL direct semantics [6, 17]—mainstream in ontology languages—offer first-order expressiveness, but they do not natively support uncertainty and decisions (*i.e.* support for efficiently representing and treating actions, values and preferences of decision makers). PR-OWL, being an extension of OWL, also offers first-order expressiveness, and it also offers support for uncertainty, but it lacks support for decisions. It was already stated that IDs offer support for decision and uncertainty, but have only propositional expressiveness. It thus becomes of interest to extend the results we have for the propositional cases to the first-order case. **Therefore, there is a need to extend the syntax of PR-OWL and its underlying logic—Multi-Entity Bayesian Network (MEBN)** [18]**—to include elements of IDs. *PR-OWL Decision*, the extension proposed in this work, addresses this issue.**

Reuse receives special attention, because it is a common, yet powerful way to drastically reduce the development effort. This is why special care is taken for backward and forward compatibility (with OWL). Backward compatibility can be achieved by designing the new language so that systems meant for the new language will automatically function with the older language, due to syntactical similarities. This offers incentives

for legacy system users to migrate to new solutions. Forward compatibility can be achieved by composing the new language's syntax with valid constructions of the older language. Legacy systems may not be able to handle the new portions perfectly, but it ought to be guaranteed that the new construction will not cause legacy systems to fail catastrophically. This increases the practical usefulness of a new solution, because part of new models can be built on well tested legacy systems.

Examples of kinds of decision problems (and related tasks) that could particularly benefit from the new solution are:

- Those which the number of decisions and available actions (choices) are not known in advance. For instance, we can have decisions that repeat over time and the number of choices may increase/decrease for each decision. Other types of repetitions (in probabilistic dependency, or on utility functions) can also be treated by PR-OWL Decision.
- Those using abstractions/concretizations from OWL class hierarchy. For instance, an OWL ontology may indicate that a "Tablet" is a subclass of "Computer", thus a decision making model developed for a "Computer" might work well with a "Tablet" (*e.g.* decision models about information theft involving computers/tablets). PR-OWL Decision handles such inheritance natively.
- When the process involving decision making itself is performed or aided by multiple software systems, interoperability plays a major role. OWL has strong support for interoperability, so does PR-OWL Decision.
- Iterative/incremental model development process may benefit from PR-OWL Decision, due to its aim in reuse. A PR-OWL Decision ontology can be developed incrementally, starting from a well-tested deterministic ontology, then creating a PR-OWL ontology which imports the deterministic ontology (so that the original ontology is kept unchanged), and finally a PR-OWL Decision ontology can import the PR-OWL ontology. Cost of verification and validation is reduced, because previously tested artifacts are reused in "as-is" basis. An example in Software Product Line domain is discussed in the following sub-section.

A. *Software Product Line (SPL) Domain*

Examples presented throughout this paper are based on a Software Product Line (SPL) ontology, which was developed as a Proof of Concept for PR-OWL Decision [19]. SPL is a "family" of software-intensive systems that share a common set of characteristics satisfying specific needs of a particular domain, and are developed from a common set of software assets [20]. The engineering process of SPL is often divided into two phases: *domain engineering* (the process of analyzing, architecting and developing reusable components among the family) and *application engineering* (process of producing a single product by integrating and/or customizing reusable components). Proper SPL practices enable fast production and customization.

Quickly developing a series of configurable/customizable software systems is important not only because software is ubiquitous in any current intelligence, defense or security system, but also because such systems are becoming increasingly complex and competitive, both in terms of pricing and available functionalities. Problems in intelligence, defense, and security are diverse, thus it's natural to think that not all clients will use of the entire set of available system features. Quickly—and automatically—offering a proper set of features to the client, given their particular needs, would help in establishing a competitive price, and also to avoid unnecessary use of computational resources caused by unused features (the latter may become rather critical in embedded systems). Our Proof of Concept model mainly addresses this issue.

The following list summarizes some important concepts of SPL that are referenced throughout this paper:

- ***Features*** are common and variant characteristics among a set of software systems. These are related to (or originated from) a set of domain requirements, and can be mapped to a set of software assets, so it can be thought as an abstraction that maps requirements to reusable components.
- ***Configuration*** can be thought as a set of features which jointly satisfies constraints of consistency (*e.g.* dependency and compatibility). We can move from a configuration to another by adding, removing, or substituting features, of course, without breaking consistency rules.
- ***Domain requirements*** are requirements identified and treated in the domain engineering process (*i.e.* "inter-system" requirements that will derive features and related reusable components).
- ***Application requirements*** are requirements treated in the application engineering process (*i.e.* emerging requirements that will result in a single product). A "requirement" in SPL can be either a domain or application requirement.

The Proof of Concept ontology was developed in a iterative/evolving manner, starting from a simple, deterministic OWL ontology, which captured the features and their constraints. Then, a PR-OWL ontology which encodes some probabilistic relationships between the features, requirements, and assets was developed by reusing (importing) the original ontology. Finally, a PR-OWL Decision ontology was developed in order to represent the costs and profits (with associated risks) of incorporating new features to some configuration given emerging requirements. The resulting ontology is able to solve, for example, a decision problem of choosing the set of features to (re)use during application engineering, under maximum expected profit (or minimum expected cost) criteria.

II. PR-OWL

Traditional ontologies have no built-in mechanism for representing or drawing inferences under uncertainty. The *Probabilistic Web Ontology Language* (PR-OWL) consists of a set of classes and properties (relationships) that collectively form a framework for building and reasoning with probabilistic ontologies, yet keeping syntactical compatibility with OWL. The purpose of a probabilistic ontology is to describe knowledge about a domain and its associated uncertainty in a principled, structured, and sharable way, so that it can be applied to support semantic applications working in complex open-world environments. PR-OWL 2 is an extension of OWL 2 with

enhanced meta-level [1] support for specifying probability distributions of OWL properties [8]. Constructs of PR-OWL basically follow an abstraction inherent from *Multi-Entity Bayesian network*, which is explained in next sub-section.

A. *Multi-Entity Bayesian Network*

Multi-Entity Bayesian Network (MEBN) [18] is the underlying logic of PR-OWL (and its version 2). For this reason, a PR-OWL specification can be informally seen as a scheme for describing a MEBN model in OWL.

MEBN extends BN [21] by combining the expressiveness of First-Order Logic and the inference power of BN. MEBN represents the world as a collection of inter-related entities, their respective attributes, and relations among them. Knowledge about attributes of entities and their relationships is represented as a collection of repeatable patterns, known as MEBN Fragments (MFrags). A set of well-defined MFrags that collectively satisfies first-order logical constraints ensuring a unique joint probability distribution is a MEBN Theory (MTheory). The probabilistic portion of a consistent PR-OWL 2 ontology represents an MTheory.

An MFrag represents uncertain knowledge about a collection of related random variables (RVs). RVs, also known as "nodes" of an MFrag, represent the attributes and properties of a set of entities. A directed graph represents dependencies among the RVs. Since an MFrag is in fact a template that can be repeatedly instantiated to form *Situation-Specific Bayesian Networks* (SSBNs), their RVs usually contain as arguments one or more ordinary variables, which are variables that are substituted by instances of entities during the instantiation process. SSBNs are regular BNs that are formed, usually in response to a query, to address a particular situation that may occur in the domain. Since a SSBN is just a regular BN, traditional BN algorithms, like junction tree algorithm [22], can be applied to it with no special adaptations. Usually, a SSBN would look like a collection of "similar" nodes, differing only by their arguments' values.

MEBN provides a compact way to represent repeated structures in a Bayesian Network. An important advantage of MEBN is that there is no fixed limit on the number of random variable instances, which can be dynamically instantiated as needed. Some may see MFrags as tiny "chunks of knowledge" of a given domain. Since a MTheory is a consistent composition of such "chunks", MEBN (as a formalism) is suitable for use cases addressing reuse of information. This property is used in this work in order to achieve efficient reuse of ontology.

Finally, MEBN categorizes random variables into three different types. See Figure Fig 1 for a graphical representation. Directed arrows going from parent to child variables represent dependencies. The list of arguments in parenthesis are replaced by unique individuals when the SSBN instantiation process is triggered. The following list describes the elements presented in Fig 1:

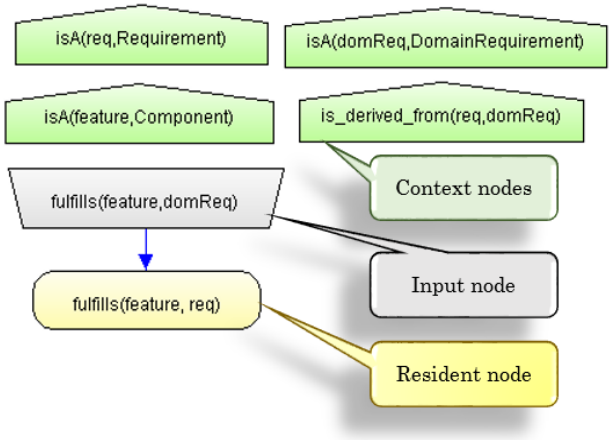

Fig 1. Structure of MEBN Fragment.

- *Resident nodes* (rounded yellow rectangles) are predicates (as in First-Order Logic) which represent the actual random variables that form the core subject of an MFrag. MEBN logic requires that the local probabilistic distribution of each resident node should be uniquely and explicitly defined in its home MFrag. The possible values of a resident node can be instances of entities (*e.g.* individuals of an OWL class). In this example, the resident node "*fulfills*" represents a relationship between a feature and a set of requirements (of any type) that the feature satisfies/fulfills.
- *Context nodes* (green pentagons) are Boolean (*i.e.* logical datatype) random variables representing conditions that must be satisfied to make a distribution in an MFrag valid. First-Order Logic formula (which may reference predicates in other MFrags) can be used in order to express complex conditions. For instance, the context node *is_derived_from(req,domReq)* indicates that the MFrag is only valid if *req* (a requirement) is derived from *domReq* (a domain requirement). Any combination of *req* and *domReq* not satisfying the context node will cause the instances of the nodes in that MFrag to be marked as invalid and thus some default probability distribution (instead of the distribution specified in the MFrag) will be applied.
- *Input nodes* (grey trapezoids) are basically "pointers" referencing to some resident node. Input nodes also provide a mechanism to allow resident nodes' re-usage between MFrags. In the example, the input node *fulfills(feature, domReq)* is a reference to the resident node *fulfills* in the same MFrag. The arc from *fulfills* input node to *fulfills* resident node (*i.e.* the recursive dependency) indicates that whether a feature fulfills or not some requirement depends on whether the feature fulfills or not a domain requirement which derived the requirement in question.
- *Ordinary variables* appear as arguments of nodes in the example (see labels *feature*, *req*, and *domReq*). They are "non-random" variables that can be replaced with instances

[1] The language offers means for specifying or extending information or rules about other elements in the ontology.

of entities in order to fill the arguments of nodes. Constraints about the type of ordinary variables are declared in "*isA*" context nodes, whose first argument is an ordinary variable and the second argument is a name of some entity (*e.g.* some OWL class).

III. Multi-Entity Decision Graph

Multi-Entity Decision Graph (MEDG) provides a framework for modeling and solving decision problems which require both first-order expressiveness and handling of uncertainty; and it forms the semantics, mathematical formalism, and a graphical abstraction of documents written in PR-OWL Decision. Consequently, in a technical view, PR-OWL Decision documents can be seen as a computer-readable representation of MEDG models that can be persisted in storage media or streamed to a network.

MEDG extends MEBN by combining the expressiveness of a probabilistic First-Order Logic—MEBN—with the ability to represent decisions and values (utilities) and to perform decision making under uncertainty, with maximum expected utility criterion, of Influence Diagrams (ID) [16]. IDs are a generalization of Bayesian Networks (BN) [21] which consist of a directed acyclic graph of *probabilistic nodes* (just like nodes in BN, it corresponds to random variables), *decision nodes* (they correspond to decisions to be made, and represent available actions), *utility nodes* (corresponds to utility functions, which quantifies values or preferences of a decision maker), *conditional arcs* (arcs that points to a probabilistic node and represent probabilistic dependence), *information arcs* (arcs that points to decision nodes and represent information that have to be available at the time of the decision), and *functional arcs* (arcs that points to utility nodes and represent inputs for the utility function). The main idea of MEDG is, therefore, to augment MEBN with decision nodes, utility nodes, information arcs and functional arcs.

Following the convention of MEBN, the world is represented in MEDG as a collection of inter-related entities, their respective attributes, and relations among them. Knowledge about attributes of entities and their relationships is represented as a collection of network fragments that represent repeatable patterns, known as *MFrags* (now, this name stands for *MEDG Fragments* instead of *MEBN Fragments*). A set of well-defined MFrags that collectively satisfies logical constraints is called *MTheory* (similarly, this name now stands for *MEDG Theory*). A consistent PR-OWL Decision ontology represents an MTheory. Fig 2 shows the components of a MEDG Fragment, and the following list is a description of such components:

- **Decision resident node**: this orange rectangular node is a new type of node in MEDG and it represents the class of decision nodes. It can be used in input nodes or context nodes, and just like resident nodes it needs to be uniquely and explicitly defined in some home MFrag. As in IDs, arcs pointing to these nodes are information arcs that represent information that are assumed to be known at the time of taking the action. In the example, *incorporateFeature* represents the decision of whether to add or not some feature "*feat*" to the current configuration "*config*", given

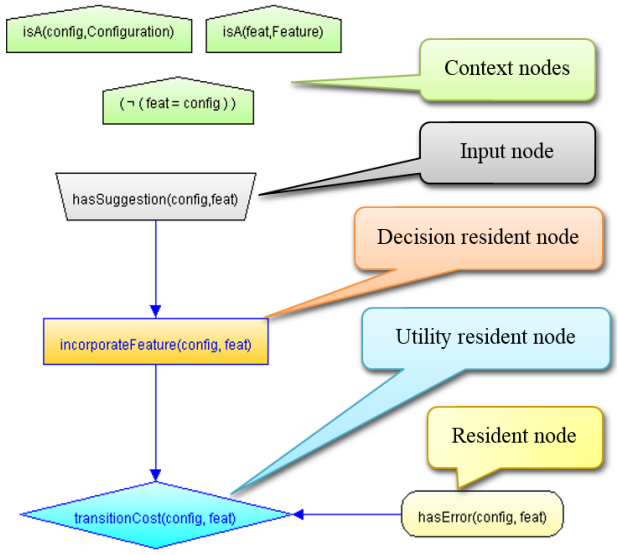

Fig 2. Structure of MEDG Fragment.

information of *hasSuggestion* (whether such feature can be suggested to the configuration or not).

- **Utility resident node**: this blue diamond node is a new type of node in MEDG which represents the class of utility nodes. MEDG logic requires that the utility function of a utility resident node must be uniquely and explicitly defined in some home MFrag. Utility resident nodes cannot be parents of resident nodes or decision resident nodes, and cannot be used in context nodes. Arcs pointing to these nodes are functional arcs and represent inputs of the utility function. Under the multi-attribute utility criteria, we can represent the "global" utility function as a combination of sub-functions (*i.e.* the utility function can be decomposed to multiple sub-functions involving only a smaller subset of variables, and each of such sub-functions can be represented by utility resident nodes). In such context, when some utility resident node is a child of utility resident nodes, it represents the combining function over the parents. If no such combining function is specified, then the unweighted additive function (*i.e.* a simple sum over the sub-functions) is implicitly assumed by default. In Fig 2, *transitionCost* represents the cost of adding the feature "*feat*" to the current configuration "*config*" (given the decision about whether to actually add or not such feature).

- **Resident node** (or "probabilistic" resident node), **input node**, **context node**, and **ordinary variables**: these elements play the same role as in MEBN. However, input and context nodes can now have references to Decision resident nodes. The three context nodes in Fig 2 are declaring that the type of the ordinary variable *config* and *feat* are respectively the *Configuration* and *Feature* entities, and the values of these ordinary variables must not be equal. The input node *hasSuggestion* is a reference to a resident node in another MFrag (not shown in the figure, though). The resident node *hasError* is the probability of the new feature *feat* to cause error to current configuration *config*, and it has direct impact on the utility.

From a semantic viewpoint, backward compatibility (*i.e.* tools that support MEDG should also support MEBN) is only possible if MEDG models without presence of decision and utility nodes are equivalent to the respective MEBN model. This explains why components of MEBN (*e.g.* resident nodes, input nodes, context nodes) are fully reused in MEDG. It is worth noting that these approaches for backward compatibility are directly applicable to PR-OWL Decision as well, because PR-OWL Decision ontologies semantically represent MEDG models, and they share the same abstractions (*i.e.* nodes, entities, states, *etc*.).

On the other hand, forward compatibility (*i.e.* tools that support MEBN should be able to open MEDG models) is not directly guaranteed at the logic level, obviously because MEBN semantics cannot handle decision and utility nodes. Instead, forward compatibility is achieved at the syntactical level in PR-OWL Decision by asserting that decision resident nodes and utility resident nodes in PR-OWL Decision are subclasses of resident nodes of PR-OWL. This shall enable tools compatible with PR-OWL to open PR-OWL Decision ontologies, and allow decision and utility nodes to be displayed and edited as if they were just resident nodes. This is why decision resident nodes and utility resident nodes in PR-OWL Decision are defined respectively as resident nodes with no probability distribution, and single-valued resident nodes in PR-OWL Decision.

A. Entailments of PR-OWL Decision: MEDG Inference

Entailments of PR-OWL Decision are information that can be inferred from a PR-OWL Decision ontology document, based on its underlying semantics—MEDG. This includes anything that can be deterministically inferred (by First-Order Logic or its subsets), anything that can be inferred by first-order probabilistic reasoning (which requires combination of First-Order Logic and probabilistic inference), and anything that can be inferred by combining the previous inference with decisions and utility functions. The former two can be achieved with MEBN and PR-OWL (actually, the first one can even be achieved with OWL direct semantics and description logic reasoning), so they are not important in the context of this document. The last one is our focus, because it requires inference in MEDG semantics.

Namely, the tasks of calculating expected utility, and to find optimal policy under maximum expected utility criterion are important entailments of PR-OWL Decision that will be considered in this research. We propose an algorithm (described in Listing 1) adapted from [23] for grounding a MEDG Theory based on entity information and evidence currently available in the knowledge/data base (in the context of PR-OWL Decision, the knowledge/data base is the ontology itself, or it can be a separate ontology, but consistent with PR-OWL Decision) to generate a Situation-Specific Influence Diagram (SSID) in order to solve the above tasks.

Fig 3 illustrates grounded inference of MEDG in the context of PR-OWL Decision. In the figure, data/evidences retrieved without probabilistic inference (*e.g.* OWL individuals or OWL property assertions) will be combined with elements of MEDG in order to instantiate the SSID. Once SSIDs are generated, they are equal to ordinary IDs, so any algorithm for solving (*e.g.* calculate expected utility or find optimal policy) IDs can be used to solve SSIDs.

Inputs:
- Queries: a list of nodes (instances of decision or resident nodes) that will be guaranteed to be present in SSID.
- Instances of entities: collection of all known instances of entities. These can be OWL individuals in PR-OWL Decision.
- Evidence: list of all random variables and decision nodes with known values (and their respective values as well).

1 Include all nodes in evidence and queries in SSID.
2 Include all possible instantiations of utility nodes (by instantiating all possible values of arguments of utility resident nodes) to SSID.
3 Mark all nodes in SSID as "unfinished".
4 For each "unfinished" node "n" in SSID, do:
  4.1 Find the resident node (or decision/utility resident node) "res" whose "n" is its instance.
  4.2 If the MFrag of "res" is marked as "unsatisfiable", set "n" to use default distribution, mark "n" as "finished", and continue at line 4.
  4.3 For each context node "cx" in the same MFrag
    4.3.1 If "cx" is unsatisfiable (*i.e.* 100% false), then mark the MFrag as "unsatisfiable", set "n" to use default distribution, mark "n" as "finished", and continue at line 4.
    4.3.2 Else if "cx" is unknown (*i.e.* neither 100% true or 100% false), then:
      4.3.2.1 Virtually transform the context "cx" to input node.
      4.3.2.2 Create arcs from new input node to all resident nodes (and decision nodes) in same MFrag.
  4.4 For each parent "p_res" of "res", do:
    4.4.1 Instantiate arguments (ordinary variables) of "p_res" that match the formulae in context nodes in the same MFrag.
    4.4.2 Instantiate "p_res" with the combination of arguments found in previous step.
    4.4.3 For each instance "p_n" of "p_res", do:
      4.4.3.1 Mark "p_n" as "unfinished", and add it to SSID (if not already there).
      4.4.3.2 Add arcs from "p_n" to "n" in the SSID.
  4.5 Mark "n" as "finished".
5 Prune (remove) from SSID all nodes that are d-separated or disconnected from queries and utility nodes.
6 Compile the LPD/utility scripts of all probabilistic and utility nodes, so that the scripts are translated to actual probability distributions/tables or actual utility functions/tables.
7 Return (output) SSID.

Listing 1: pseudocode for generating SSID.

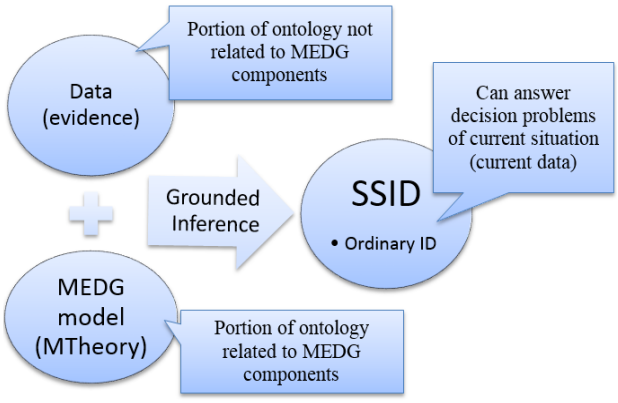

Fig 3. Grounded inference of MEDG.

B. A Script Language for Utility and Probability Distribution

A resident node in MEDG specifies a *Local Probability Distribution* (LPD), a generic specification of conditional probabilities of random variables that can be instantiated from that resident node, given their parents. However, since MEDG represents generalizations, LPDs cannot be specified in a "propositional" manner, like a table of conditional probabilities for all possible combinations of parents' states. Similarly, utility functions of utility resident nodes also cannot be specified in a "propositional" manner.

We propose a scripting language for specifying LPDs and utility functions in MEDG in a uniform and "non-propositional" manner, by extending the scripting language of [23, pp. 17-18] with more support for first-order syntax, such as support for ordinary variables in conditions, support for arguments in nodes, more support for nodes with states dynamically instantiated, and support for non-normalized values (for utilities, which do not necessarily sum up to 1). Special care was taken for backward compatibility, so that old scripts are also valid in the new grammar.

Listing 2 shows a tentative version of the new grammar in Backus–Naur Form [24] for a script for specifying utility and LPD. Listing 3 is an example of LPD script that complies with the proposed LPD grammar (it specifies the probability distribution of node *fulfills* of Fig 1).

Table I is an example of a conditional probability table that can be generated from Listing 3, when SSID is instantiated. In this example, the ordinary variable "*feature*" was substituted by an entity instance called "*F1*", and the ordinary variable "*domReq*" (*i.e.* a domain requirement) was substituted by entity instances "R1" and "R2". We can see in the table that if at least one parent is true, then the probabilities are set to true = 0.7, and false = 0.3. When no parent is true, but at least one parent is false, then the probabilities are set to true = 0.1, and false = 0.9. Otherwise, the probability of absurd is set to 1. This complies with Listing 3.

Scripts for specifying LPDs are not formally part of PR-OWL, so such scripts are directly stored as literal data properties. We will follow the same approach and store scripts in the new grammar as literal (text) data properties in PR-OWL Decision as well. Consequently, the new LPD scripting language is not formally a part of the specification of PR-OWL Decision.

```
<distribution> ::= <statement> | <if_statement>
<if_statement> ::=
            "if" <allop> <varsetname>
            "have" "(" <b_expression> ")" <statement>
            "else" <else_statement>
<allop> ::= "any" | "all"
<varsetname> ::= <ident>[["."|","]<ident>]*
<b_expression> ::= <b_term> [ "|" <b_term> ]*
<b_term> ::= <not_factor> [ "&" <not_factor> ]*
<not_factor> ::= [ "~" ] <b_factor>
<b_factor> ::= "(" <b_expression> ")"
            | <ident> ["(" <arguments> ")"]
            "=" <ident> ["(" <arguments> ")"]
<arguments> ::= <ident>[["."|","]<ident>]*
<else_statement> ::= <statement> | <if_statement>
<statement> ::= "[" <assignment_or_if> "]"
<assignment_or_if> ::= <assignment> | <if_statement>
<assignment> ::= <ident> "=" <expression> [ "," <assignment> ]*
<expression> ::= <term> [ <addop> <term> ]*
<term> ::= <signed_factor> [ <mulop> <signed_factor> ]*
<signed_factor> ::= [ <addop> ] <factor>
<factor> ::= <number> | <function> | "(" <expression> ")"
<function> ::= <possibleVal>
            | "CARDINALITY" "(" [<varsetname>] ")"
            | "MIN" "(" <expression> ";" <expression>")"
            | "MAX" "(" <expression>";" <expression>")"
            | <external_function>
<possibleVal> ::= <ident>
<addop> ::= "+" | "-"
<mulop> ::= "*" | "/"
<ident> ::= <letter> [ <letter> | <digit> ]*
```

Listing 2: BNF grammar of LPD/utility script.

```
if any feature,domReq have ( fulfills(feature,domReq) = true ) [
  true = .7, false = .3
] else if any feature,domReq have ( fulfills = false ) [
  true = 0.1, false = 0.9
] else [ absurd = 1 ]
```

Listing 3: Example of LPD script.

IV. PR-OWL Decision

PR-OWL Decision, the language proposed in this research, extends PR-OWL in order to support decision variables (*i.e.* actions that a decision maker can take) and utility variables (*i.e.* values and preferences) in probabilistic ontologies.

The new language provides definitions of special classes and properties (relationships) that collectively form a framework for building and reasoning with decision problems expressed as probabilistic ontologies. These new components are defined in terms of existing PR-OWL and OWL elements, so that syntactical compatibility with PR-OWL (and OWL) is achieved. In this chapter we define such new components and how they relate to PR-OWL and OWL.

We primarily extend PR-OWL version 2 (PR-OWL 2), because it offers enhanced meta-level features—not present in version 1—that allows us to represent probability distributions

TABLE I. EXAMPLE OF CONDITIONAL PROBABILITY TABLE THAT CAN BE OBTAINED FROM SCRIPT IN LISTING 3.

Parents	fulfills (F1, R1)	true			false			absurd		
	fulfills (F1, R2)	true	false	absurd	true	false	absurd	true	false	absurd
Child's states	true	0.7	0.7	0.7	0.7	0.1	0.1	0.7	0.1	0
	false	0.3	0.3	0.3	0.3	0.9	0.9	0.3	0.9	0
	absurd	0	0	0	0	0	0	0	0	1

of existing OWL properties [8]. These features are necessary conditions for semantic-level compatibility with OWL, because they enable entailments of OWL ontologies to be also contained in the entailments of PR-OWL 2. We also offer an alternative extension of PR-OWL version 1 (PR-OWL 1) for decision support in ontologies originally written in this older version as well. However, this is only kept for backward compatibility, and is superseded by the extension of PR-OWL 2. The version of PR-OWL Decision which extends PR-OWL 2 is called PR-OWL 2 Decision, and the version that extends PR-OWL 1 is called PR-OWL 1 Decision; but for simplicity, in this document We'll simply use "PR-OWL Decision" to refer to the one that extends PR-OWL 2.

A. PR-OWL Decision Schema Vocabulary

Just like any OWL and PR-OWL document, a PR-OWL Decision document needs to be built by combining a set of pre-defined building blocks. A PR-OWL Decision document is said to be syntactically valid if the document is validated against a schema vocabulary. A schema vocabulary is a document that partially defines another document's structure with a list of legal elements, attributes, built-in classes and properties.

Fig 4 illustrates how the PR-OWL Decision schema vocabulary relates to other vocabularies. The vocabulary (schema) files of PR-OWL 1 Decision and PR-OWL 2 Decision reuses constructs from PR-OWL 1 and PR-OWL 2 respectively. While the vocabularies of PR-OWL are valid ontologies in OWL direct semantics (thus, we can use the OWL "import" mechanism to reuse the entire document), the OWL RDF/XML syntax vocabulary file/document has some constructs that are not defined in OWL direct semantics, so only a subset of OWL vocabulary document is used in PR-OWL vocabulary. Finally, as the name implies, the OWL RDF/XML syntax document combines syntaxes from XML (and XML Schema) and Resource Description Framework (RDF) and its schema (RDFS) [25].

From the foundation of OWL, any ontology component is identified by an Internationalized Resource Identifier (IRI), a standard defined by the Internet Engineering Task Force to extend the Uniform Resource Identifier (URI) scheme. URIs and IRIs are both text identifiers that resemble web addresses, but URIs are limited to ASCII characters, while IRIs allow Unicode characters to be used. The stereotype <<*owl:imports*>> in arcs represents a property that is used for importing other OWL ontologies entirely. The World Wide Web Consortium (W3C) recommends not to import the OWL schema vocabulary directly to ontologies using direct semantics of OWL, because it will break some compatibility with Description Logic. Therefore, the stereotype <<*uses*>> indicates that only a subset of features are referenced. The stereotype <<*Definition*>> is used instead of <<*Vocabulary*>> in XML Schema Definition (XSD) simply because the word "definition" is part of its official name.

In the syntax viewpoint, backward compatibility with PR-OWL is forced because we explicitly import the PR-OWL schema vocabulary into the new schema (thus, tools compatible with PR-OWL Decision are forced to handle PR-OWL schema as well). Forward compatibility (*i.e.* tools compatible with OWL or PR-OWL will be able to open PR-OWL Decision documents—but not necessarily execute some reasoning process) is achieved because PR-OWL Decision schema vocabulary only uses building blocks of OWL and PR-OWL, and the PR-OWL schema vocabulary only uses building blocks compatible with OWL's RDF/XML syntax and vocabulary—thus the entire import closure is forward compatible.

B. Syntactical Differences with PR-OWL

PR-OWL Decision introduces the concept of decision nodes and utility nodes to PR-OWL. No changes will be made to existing syntactical blocks of PR-OWL, which will be fully reused—imported—by the PR-OWL Decision. Fig 5 illustrates the classes of PR-OWL 2 Decision and their relationships to PR-OWL 2 classes. Fig 6 illustrates the classes of PR-OWL 1 Decision and their relationships to PR-OWL 1 classes. The remaining paragraphs of this section basically discusses about the contents of the figures.

The prefixes of IRIs of classes in PR-OWL 2 Decision are the IRIs of its schema vocabulary (*i.e.* IRIs of these classes starts with the IRI of the schema vocabulary of PR-OWL 2 Decision, and the IRI fragment—suffix after "#"—is the name of the class). Similarly, prefixes of IRIs of classes in PR-OWL 1 Decision are the IRIs of the schema vocabulary of PR-OWL 1 Decision. For example, the IRI of class *DomainDecisionNode* of PR-OWL 2 Decision is <*http://www.pr-owl.org/pr-owl2-*

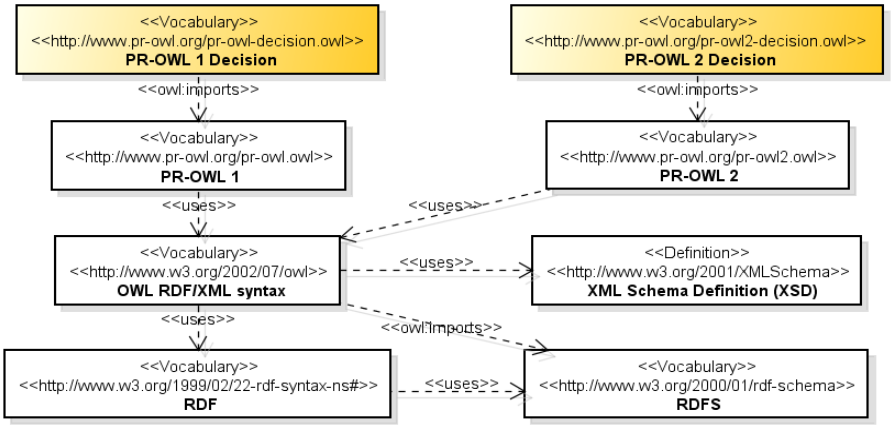

Fig 4. IRI/URI of vocabularies.

decision.owl#DomainDecisionNode>. The IRIs of the other classes follow the same pattern, so IRIs are omitted in the figures for sake of visibility. These classes can be mapped to components of its underlying logic—Multi-Entity Decision Graph (MEDG)—which is presented in later section.

The following list describes the main elements of PR-OWL 2 Decision in Fig 5—again, please refer to the section about Multi-Entity Decision Graph for the semantics of these elements:

- *DomainDecisionNode*: this class represents decision resident nodes of MEDG (see next section for descriptions about MEDG). It extends *DomainResidentNode*, a class which represents resident nodes in PR-OWL, because all properties that are valid for the *DomainResidentNode* (for instance, it should be associated with possible values, can have parents and children, and can be used as arguments of other nodes) are also valid for *DomainDecisionNode*, except for the fact that LPDs are not used in *DomainDecisionNode*.
- *DomainUtilityNode*: this class represents utility functions (utility resident nodes in MEDG). This is represented as subclass of *DomainResidentNode* for forward compatibility, so that tools compatible with PR-OWL can open utility nodes as if they were resident nodes with a single possible value (the *utility* instance).
- *UtilityMExpression*: this is an extension of *MExpression* of PR-OWL 2 for *DomainUtilityNode*. The *MExpression* connects *Node* to its arguments, types, or possible values, and *UtilityMExpression* specifies some restrictions that force *DomainUtilityNode* not to be used in arguments of context nodes (this is achieved by "*isTypeOfArgumentIn exactly 0 MExpressionArgument*" restriction), and by forcing the type of *DomainUtilityNode* to be always *UtilityVariable*.
- *UtilityVariable*: this is an extension of *RandomVariable*, a class which describes the type of *MExpression*. *UtilityVariable* is used to force *DomainUtilityNode* to be associated with only a single possible value: the *utility*. This asserts that tools compatible with PR-OWL will see instances of *DomainUtilityNode* as being resident nodes with a single value.
- *utility*: this OWL individual is a possible state of *DomainUtilityNode* created for compatibility with PR-OWL (thus, this OWL individual does not actually represent the "concept" of utility), because constraints in PR-OWL forces any node to have at least one possible state. Please, notice that numerical values of utilities in PR-OWL Decision are represented in terms of utility functions, not by some OWL individual or literal called "utility". This is similar to the approach in PR-OWL for probabilities, because such values are represented as probability distributions, not by some individual or literal called "probability".

The following list describes the elements of PR-OWL 1 Decision (Fig 6) and compares them with PR-OWL 2 Decision:

- *Domain_Decision*: same of *DomainDecisionNode* of PR-OWL 2 Decision.
- *Domain_Utility*: same of *DomainUtilityNode* of PR-OWL 2 Decision. The "*isArgTermIn exactly 0 ArgRelationship*" forces *Domain_Utility* not to be used as arguments in context

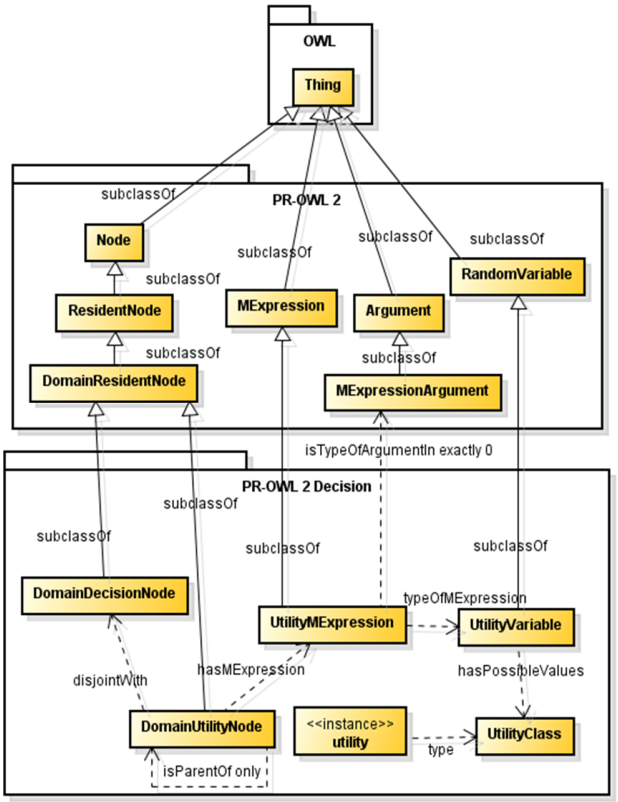

Fig 5. PR-OWL 2 Decision classes and relations to PR-OWL 2.

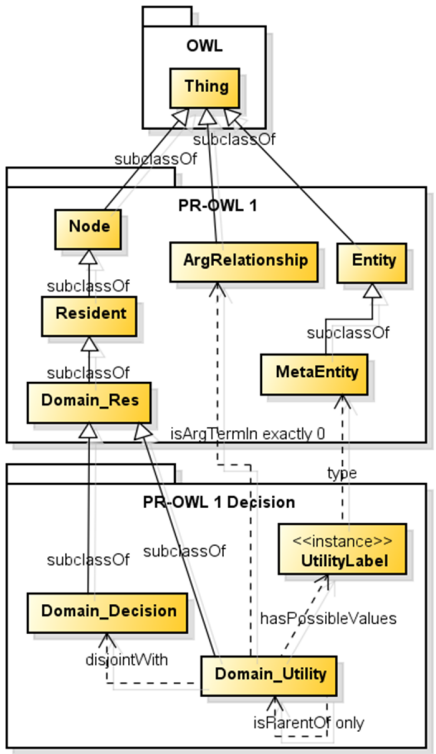

Fig 6. PR-OWL 1 Decision classes and relations to PR-OWL 1.

nodes; and the restriction "*hasPossibleValues utility*" enables tools compatible with PR-OWL 1 to see *Domain_Utility* as a resident node with single value.

- *UtilityLabel*: this is the same of *utility* in PR-OWL 2 Decision. The difference is that a constraint in PR-OWL 1 forces a possible state of a node to be an individual of *Entity*, while in PR-OWL 2 this constraint is relaxed. For this reason, utility in PR-OWL 1 Decision is an individual of *MetaEntity*—subclass of *Entity*.

V. Conclusion and Future Work

PR-OWL Decision was formulated as an extension to PR-OWL in order to support decision making under uncertainty. Backward and forward compatibility was ensured by reusing both syntax and semantic elements from PR-OWL. MEDG, the underlying logic of PR-OWL Decision, augments MEBN with decision and utility variables, so that entailments of PR-OWL Decision can be obtained with MEDG inference. An example of grounded inference/solving algorithm and a script for specifying probabilities and utilities in MEDG was described in this document. This work is part of an ongoing Ph.D. research, thus further details on MEDG, related algorithms, and software implementations will be coming in future works.

Acknowledgment

We thank UnBBayes development team of Universidade de Brasília, especially professor Marcelo Ladeira, M.S. student Laécio Santos, undergraduate students Guilherme Torres, Diego Marques, Rafael Martins, and Pedro Abreu for their insights and assistance with software development.

References

[1] R. N. Carvalho, P. C. G. Costa, K. B. Laskey and K. C. Chang, "PROGNOS: predictive situational awareness with probabilistic ontologies," in *In Proceedings of the 13th International Conference on Information Fusion*, Edinburgh, UK, July 2010.

[2] P. Mitra, N. F. Noy and A. R. Jaiswal, "Omen: A probabilistic ontology mapping tool," in *In The Semantic Web–ISWC 2005*, 2005.

[3] T. Tudorache, "Employing ontologies for an improved development process in collaborative engineering," 2006.

[4] H. H. Wang, Y. F. Li, J. Sun, H. Zhang and J. Pan, "Verifying feature models using OWL," *In Journal of Web Semantics,* vol. 5, no. 2, p. 117–129, June 2007.

[5] O. Udrea, D. Yu, E. Hung and V. S. Subrahmanian, "Probabilistic ontologies and relational databases," in *On the Move to Meaningful Internet Systems*, 2005.

[6] J. Carroll, I. Herman and P. F. Patel-Schneider, "OWL 2 Web Ontology Language (Second Edition)," 11 December 2012. [Online]. Available: https://www.w3.org/TR/owl2-rdf-based-semantics/. [Accessed 20 July 2016].

[7] P. C. G. Costa, "Bayesian Semantics for the Semantic Web," 2005.

[8] R. N. Carvalho, K. B. Laskey and P. C. G. Costa, "PR-OWL 2.0-bridging the gap to OWL semantics," in *In Proceedings of the 6th International Conference on Uncertainty Reasoning for the Semantic Web*, November 2010.

[9] E. Acar, C. Thorne and H. Stuckenschmidt, "Towards Decision Making via Expressive Probabilistic Ontologies," in *In Proceedings of the 4th International Conference, ADT 2015*, Lexington, KY, USA, 2015.

[10] C. Guestrin, D. Koller, C. Gearhart and N. Kanodia, "Generalizing plans to new environments in relational MDPs," in *In Proceedings of the 18th international joint conference on Artificial intelligence*, 2003.

[11] S. Joshi, K. Kersting and R. Khardon, "Generalized First Order Decision Diagrams for First Order Markov Decision Processes," in *In International Joint Conference on Artificial Intelligence*, 2009.

[12] S. Sanner, "Relational dynamic influence diagram language (RDDL): Language description," Australian National University, 2010.

[13] C. Wang, S. Joshi and R. Khardon, "First order decision diagrams for relational MDPs," in *Journal of Artificial Intelligence Research*, 2008.

[14] H. L. Younes and M. L. Littman, "PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects," Technical report. CMU-CS-04-162, 2004.

[15] D. Poole, "The independent choice logic for modelling multiple agents under uncertainty," *Artificial Intelligence,* vol. 94, no. 1, pp. 7-56, 1997.

[16] R. A. Howard and J. E. Matheson, "Influence diagrams," in *In Readings on the Principles and Applications of Decision Analysis II*, 1984/2005.

[17] I. Horrocks, B. Parsia and U. Sattler, "OWL 2 Web Ontology Language Direct Semantics (Second Edition)," 11 December 2012. [Online]. Available: https://www.w3.org/TR/owl2-direct-semantics/. [Accessed 20 July 2016].

[18] K. B. Laskey, "MEBN: A language for first-order Bayesian knowledge bases," *Artificial intelligence,* vol. 172, no. 2, pp. 140-178, 2008.

[19] S. Matsumoto, K. B. Laskey and P. C. G. Costa, *Probabilistic Ontologies in Domain Engineering,* Washington DC: presented at the Systems Engineering in DC Conference (SEDC), 2016.

[20] K. Pohl, G. Böckle and F. J. van Der Linden, Software product line engineering: foundations, principles and techniques, Springer Science & Business Media, 2005.

[21] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Revised second printing. Morgan Kaufmann, 1988/2014.

[22] F. Jensen, F. V. Jensen and S. L. Dittmer, "From influence diagrams to junction trees," in *In Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*, 1994.

[23] S. Matsumoto, R. N. Carvalho, P. C. Costa, K. B. Laskey, L. L. Santos and M. Ladeira, "There's No More Need to be a Night OWL: on the PR-OWL for a MEBN Tool Before Nightfall," in *Introduction to the Semantic Web: Concepts, Technologies and Applications*, G. P. C. Fung, Ed., iConceptPress, 2011, pp. 267-290.

[24] J. Backus, F. Bauer, J. Green, C. Katz, J. McCarthy, P. Naur, A. J. Perlis, H. Rutishauser, K. Sameison, B. Vauquois, J. H. Wegstein, A. van Wijngaarden and M. Woodger, "Revised report on the algorithmic language Algol 60," *The Computer Journal,* vol. 5, no. 4, pp. 349-367, 1963.

[25] P. Hayes and B. McBride, "RDF Semantics.," 10 February 2004. [Online]. Available: https://www.w3.org/TR/2004/REC-rdf-mt-20040210/. [Accessed 13 May 2016].

[26] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld and D. Wilkins, "PDDL-the planning domain definition language.," Technical report. Yale Center for Computational Vision and Control, 1998.