Business Rules Uncertainty Management with Probabilistic Relational Models

Hamza Agli, Philippe Bonnard, Christophe Gonzales, Pierre-Henri Wuillemin

Business Rules Uncertainty Management with Probabilistic Relational Models

Hamza AGLI[‡], Philippe BONNARD[‡],
Christophe GONZALES[⋆] and Pierre-Henri WUILLEMIN[⋆]

[‡] IBM France Lab, Gentilly, France
[⋆] Sorbonne Universités, UPMC Univ Paris 6, CNRS, UMR 7606 LIP6, Paris, France,
{hamza.agli,philippe.bonnard}@fr.ibm.com
{christophe.gonzales,pierre-henri.wuillemin}@lip6.fr

Abstract. Object-oriented Business Rules Management Systems (OO-BRMS) are a complex applications platform that provide tools for automating day-to-day business decisions. To allow more sophisticated and realistic decision-making, these tools must enable Business Rules (BRs) to handle uncertainties in the domain. For this purpose, several approaches have been proposed, but most of them rely on heuristic models that unfortunately have shortcomings and limitations. In this paper we present a solution allowing modern OO-BRMS to effectively integrate probabilistic reasoning for uncertainty management. This solution has a coupling approach with Probabilistic Relational Models (PRMs) and facilitates the inter-operability, hence, the separation between business and probabilistic logic. We apply our approach to an existing BRMS and discuss implications of the knowledge base dynamicity on the probabilistic inference.

Keywords: Business Rules Management Systems, Uncertainty management, Probabilistic Relational Models, Bayesian Networks

1 Introduction and related work

OO-BRMS are very popular tools for decision-making automation. They are considered as the evolution of rule-based expert systems. In a separation between application and business logic, these systems facilitate authoring, checking, deploying and executing day-to-day companies operational business policies. Indeed, business professionals and IT specialists can collaborate relatively independently using such systems. This is because they provide double level artifacts that align IT practices with business needs [3,9].
Whereas BRMS are well adapted to deal with structured and complete data using classical Boolean inference, they face difficulties when they take into account uncertain or incomplete data. To tackle the issue of uncertainties in the domain, three approaches are commonly used:

– Heuristic models to weight rules with a degree of truth, e.g., certainty factors (CF) and likelihood ratios (LR) [5,10]. These deal with uncertainty in the

knowledge (rules) not the data. However probabilistic interpretation given to CF is incoherent with probability theory [11]. On the other hand, the conditional independence between evidence and rules actions in LR is rarely satisfied in real applications and LR-based expert systems have poor performance [16].
- *Fuzzy logic* (FL) [21] to capture the uncertainty and imprecision by associating variable values to fuzzy sets, but in essence, FL is not conceived to deal with incomplete data or to express relations between variables in the knowledge base as in OO frameworks. Besides, fuzzy logic when applied to systems that performs chains of inference, such as BRMS may lead to inconsistent conclusions [7].
- Bayesian techniques, which are essentially based on Bayesian Networks (BNs) [17], to consistently model domains with uncertainty. In addition, several algorithms have been proposed to learn their graphical structure and their conditional probability tables (CPTs) parameters. Even if they are a very popular tool to deal with uncertainty, BNs are not suited for complex systems, in which they involve high design and maintenance costs [15,13]. Besides, they do not support well object-oriented and dynamic systems.

One can also find hybrid approaches that combine, for instance, BNs with CF [14,4]. Obviously these methods incur in problems discussed previously. Moreover they are developed for specific use and cannot handle effectively the frequent changes of business policies, where BRMS perform better. Another approach is Probabilistic Logic Programming [6]. But this is not suited for the BRs procedural side effects and the OO-BRMS upon which we build our application. To summarize, current BRMS uncertainty management state-of-the-art face theoretical and practical limitations, do not exploit structural information encoded in the knowledge base and face scaling difficulties.

To overcome the previous limitations, we propose to couple BRMS with Probabilistic Relational Models (PRMs) [12,18,19], an object-oriented extension of BNs that enables handling very large systems. Their object paradigm and relational model allow them to be a good candidate for managing uncertainty in an OO-BRMS. In addition, PRMs are equipped with sophisticated inference engines that enable to answer efficiently various types of probabilistic queries.

In this paper, we describe a method that allows modern OO-BRMS to reason under uncertainty using a coupling approach that separates uncertainty and rules management. There are many reasons for this separation. First, trying to manage PRM inference and update inside BRMS would be inefficient and difficult since the rule engine is non-monotonic and is by essence a procedural engine on object data. Second, separating concepts and architectures simplifies the software maintaining and offers more control over the framework complexity. Last, such a coupling gives a mathematically sound interpretation of the uncertainty and is based on a framework that is essentially designed to cope with large and complex systems. This work is the continuation of [1] that proves the feasibility and describes the coupling framework.

The remainder of this paper is organized as follows: in the next section, we

briefly introduce OO-BRMS and PRMs. Then, we present the coupling approach in Section 3, as a solution that allows OO-BRMS to deal with uncertainty. Then Section 4 describes how we implemented this solution in practice. Finally, some conclusion and future works are provided in Section 5.

2 Preliminaries

2.1 Object-oriented BRs

BRs are rules in the form "*IF condition THEN action*" that are exploited for reasoning by forward chaining inference engines. OO-BRRMS execute BRs against an object model (OM) that describes the application objects. Let us illustrate this through a simplified example from an insurance application. Assume the model consist of three classes representing a healthcare professional, a subscriber and a reimbursement request. Fig. 1 gives a UML class diagram for this application.

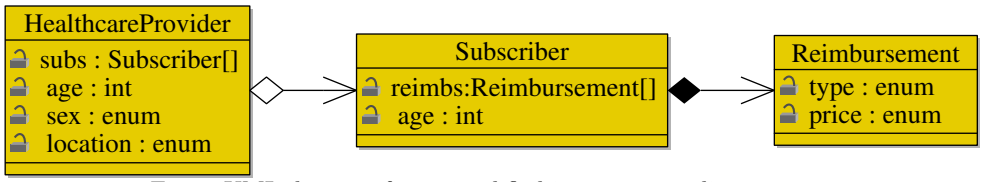

Fig. 1: UML diagram for a simplified insurance application

In a fraud detection context, we want to verify, using BRs-based approach, whether the healthcare professional is fraudulent. In such a way, anomalies that indicate fraud are detected by executing a set of rules and using scoring heuristics. For instance, if a fraud detection rule says that an excessive invoice alert must be raised on a healthcare provider who submits a high price reimbursement request for one of his subscribers, the corresponding object-oriented BR in Rule 1.1 will look for objects in the working memory (WM) that corresponds to providers with subscribers requesting reimbursements with a high price.

Rule 1.1: detect invoice anomaly

```
1 IF hp has type HealthcareProvider & sub has type Subscriber &
      reimb has type Reimbursement & sub in hp.subs & reimb in
      sub.reimbs & reimb.price == high
2 THEN raiseAlertExceededInvoicePrice(hp)
```

Similarly, another rule says that a lens age anomaly alert must be raised on a healthcare provider who submits a lens reimbursement request for a subscriber under age 10. Rule 1.2 shows its pseudo-code.

Rule 1.2: detect lens anomaly

```
IF hp has type HealthcareProvider & sub has type Subscriber &
    reimb has type Reimbursement
     & sub in hp.subs & reimb in sub.reimbs & sub.age < 10 &
        reimb.type=lens
THEN raiseAlertLensAgeAnomaly(hp)
```

When the data is completely known and well adapted to classical logic paradigm, such rules are well handled using variant of pattern matching algorithms, e.g, enhanced RETE [8] or stateless sequential execution. However, in front of uncertain or missing data, such rules cannot be executed. The next paragraph introduce theoretical foundations to handle such a situation using PRMs.

2.2 PRMs

One reason why standard BNs do not scale well is because they do not exploit the structure of the data. Instead, PRMs share the same class concept used in the object models. Indeed, the idea is that in complex systems, one can often identify repeated patterns, which can be abstracted as classes. Each pattern represents a fragment of a BN over its random variables. These correspond to the class descriptive attributes. PRMs also define the mechanism of reference slot allowing the navigation between attributes of different classes, and hence, the good definition of conditional probability distribution. They use aggregators to express many-to-one instance relations and get around the issue of multiple class definitions w.r.t variable number of configurations depending on relation arities. Finally, PRMs define a relational skeleton that represents the instance graph.

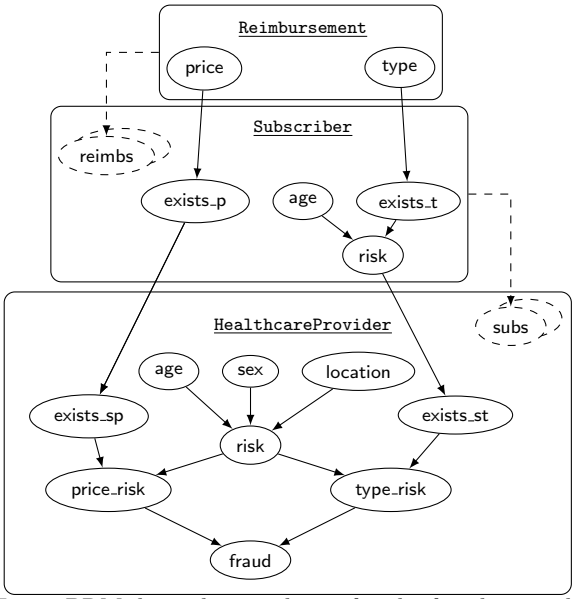

Fig. 2: PRM dependency schema for the fraud example

This corresponds to the PRM system: classes that are instantiated and linked using reference slots. To sum up, a class corresponds to a set of random variables that share common relations (abstraction of repeated patterns) and are gathered in a BN fragment. Classes communicate through reference slots.
It is easy to see that the PRM can, not only represent the object model, but also the relation or causal/influence directions between the model attributes. Given an attribute, these relations are expressed through an arc connection between this attribute and its immediate predecessors, which are called "parents" in the graphical structure. In this paper, both relations and CPTs are assumed to be provided by a domain expert or obtained from a learning process.
One possible PRM representation of the fraud detection example is showed in Fig. 2. For instance, the attribute `reimbs` of class `Subscriber` is a multiple reference slot, which shows that the class points to a set of `Reimbursement`. In the running example, a divide-and-conquer approach is used to build aggreagators: we first determine whether the `Subscriber` has a `Reimbursement` with a `high price` (by `exists_p` aggregator); second, we determine if the `HealthcareProvider` is linked to a `Subscriber` satisfying the previous condition (by `exists_sp`). We follow the same reasoning to generate the aggregator `exists_st`.
Fig. 3 depicts an example of a relational skeleton obtained from the fraud example instances. A dashed arc stands for a reference slot, for instance `sub1` references `reimb1` and `reimb2`. Further details bout the PRMs extension used

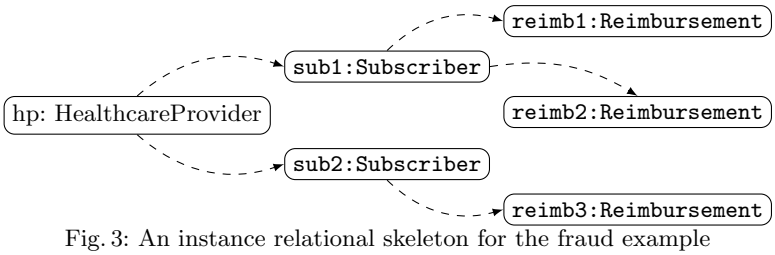

Fig. 3: An instance relational skeleton for the fraud example

in this work might be found in [20].

3 Coupling BRs and PRMs

In the previous paragraph, we highlighted the common OO paradigm that ties OO-BRMS and PRMs and we illustrated this through the fraud example. Using PRMs allows for more model abstraction, while using classical BNs methods results in a repetition of objects creation, model dependence in rules and inference inefficiency for large systems. In this paragraph, we suggest to extend the BRs data meta-model, compilation and runtime to specify relations and probability model.

3.1 Probabilistic rules

We propose to use the aforementioned similarity to invoke probabilistic instructions within the rules, e.g, marginal distribution computation and evidence posting. For this reason, probabilistic attributes in the rules are directly mapped to their equivalent PRM attributes (see Section 2). Assume that the OM against which the rules are executed is extended to include all the nodes in the dependency network on Fig. 2 as attributes of the corresponding classes. Assume further that a `prob` operator [1] is introduced in the syntax and allows triggering inference process of the probabilistic engine. As we discussed previously, PRM relates attributes of different classes, and those of generated instances consequently, to permit the creation of complex networks covering multiple instances. Although, RVs are generated from the same classes, they should be regarded as distinct variables with their own life-cycle. We know from Fig. 2 that `price_risk` attribute is linked to attributes of classes `Subscriber` and `Reimbursement` in the PRM by reference mechanism, hence, in this new extension, there is no need to evaluate conditions that can be processed by probabilistic inference. When the engine encounters the `prob` operator, it immediately launches the probabilistic process, which queries the underlying PRM. In such an extension, probabilistic data is explicitly identified and can be processed by PRM engines. Now, instead of Rule 1.1, which says basically, that an alert must be raised when a healthcare provider submits a fraudulently expensive price reimbursement for a subscriber, we can have Rule 1.3 that says that an excessive invoice alert must be raised on a healthcare provider if there is a 80% probability that the price of a reimbursement request is excessive.

Rule 1.3: detect invoice anomaly with probability

```
IF hp has type HealthcareProvider
   & prob(hp.price_risk=high)>.8
THEN raiseAlertExceededInvoicePrice(hp)
```

3.2 BRs object model extension

We suggest to extend both rules, by adding new attributes as in the previous section, and their data meta-model by adding probabilistic annotations. This has two advantages. The first is moving probabilistic definitions from rules to their data meta-model. In making this move, BRs can externalize probabilistic inference and allow for separate management of business and probabilistic logic. Second, this enables the model to be more independent w.r.t the rules, which means an independent evolution of both. Annotations are a type of meta-data that enriches the meta-model at hand. In this work, they are added to indicate that a class contains probabilistic information, as well as that an attribute is mapped to a PRM attribute and is parametrized by its corresponding CPT and parents. If the attribute is an aggregator, annotations show its type, its domain

[1] for probability

and the concerned modalities, i.e, random variables (RVs) possible states. As we can see, such annotations allow for a natural mapping between OM and PRMs. Therefore, an OM class (resp. attribute) is mapped to a PRM class (resp. attribute) and the probabilistic data and how classes are related to each other is extracted from the OM annotations. In the OM, a restricted type represents a type whose domain is restricted, for instance an `integer` that is restricted to $\{0, 1, 2\}$. Only discrete RVs are supported in PRMs, they can be user-labeled (e.g, `state_type`) or built-in types (e.g, `boolean,int`). Thanks to these annotations, rules engine can generate the underlying PRM classes and system at compile time. Before the generation process, the model is parsed and checked. For example we check if the given list of a PRM attribute parents is valid and consistent with its CPT. This latter is also checked to verify it represents a well defined probability distribution. Actually, there are two possible modes for PRM system definition. The first is a static declaration, which assumes that all WM instances are known at compile-time. The PRM system is then generated either by directly processing the WM instance graph, or by an explicit declaration inside a special annotated class, which also specifies necessary relations. The second mode allows a dynamic definition in addition to the previous mode. Here, rules execution may also update the system by incrementally inserting new instances or modifying relations for instance. The last mode is obviously much more interesting since it reflects BRs and WM dynamic nature. The mapping we use allows the rules to generate complex probabilistic networks via the simple mechanism of class instanciation and reference slot. This power property enables the rules, for instance to handle many sets of RVs, which are obtained for free, just by means of linking instantiated classes.

4 Implementation

To illustrate all the concepts introduced in previous sections, we implemented a prototype that couples IBM Operational Decision Manager (ODM)[2] as an OO-BRMS with A Graphical Universal Model (aGrUM) [3] as a probabilistic engine. However, the methodology we applied can be easily generalized to any OO-BRMS as we showed previously. ODM execute BRs against an eXecutable OM, hereafter XOM, using the Ilog Rule Language (IRL). The latter is a Java-like language, which is also based on the OO paradigm. In practice, this model can be build from Java sources for instance. The XOM is a class model that describes the application objects and data of the WM. ODM allows also business professionals to enter rules using the Business Action Language (BAL), which describes rules[4] in a more human readable format. Finally, ODM provides an automated mapping between both BAL business and IRL technical rules[5]. To begin with, let us show the IRL classes obtained for `Subscriber` and the system of our running

[2] http://www-03.ibm.com/software/products/en/odm
[3] http://agrum.lip6.fr
[4] the series of "if-then(-else)" statements
[5] actually, this automation is not always defined, but may require IT specialist insight.

example. Consider Fig. 4 line 1, the annotation `@PrmClass` is a mark to express that the class contains probabilistic information. The corresponding probabilistic attributes are annotated with `@PrmAttribute` and carry information needed to describe their counterpart PRM attributes. For instance, `age` in line 7 has no parents and a CPT describing whether the subscriber is under the age of 10 is given. Note that `AgeType` at line 7 is an `Integer` restricted type. `@PrmAgg` marks the attribute as an aggregator. In line 15, the annotation specifies a list of the attribute parents and its CPT. In this example, we implemented the static mode. So, instances are specified as internal attributes of the system class that is annotated with `@PrmSystemClass` in line 19. Reference slots are set inside the class constructor at line 25. Finally, the relational skeleton in Fig. 3 is generated from this system class.

```
1  @PrmClass
2   public class Subscriber{
3    @PrmMultiReference
4    public Reimbursement[] reimbs;
5
6    @PrmAttribute(parents={},cpt={{.2},{.8}})
7    public AgeType age;
8
9    @PrmAgg(name="exists",attribute="reimbs.type",mod="lens")
10   public boolean exists_t;
11
12   @PrmAgg(name="exists",attribute="reimbs.price",mod="high")
13   public boolean exists_p;
14
15   @PrmAttribute(parents={"age","exists_t"},cpt
         ={{.2,.6,.5,.3},{.8,.4,.5,.7}})
16   public boolean risk;
17 }
18
19 @PrmSystemClass
20  public class System{
21   public HealthcareProvider hp = new HealthcareProvider();
22   public Reimbursement[] reimbs  = new Reimbursement[3];
23   public Subscriber[] subs = new Subscriber[2];
24
25   public System(){
26    hp.subs=subs;
27    sub[0].reimbs={reimbs[0],reimbs[1]};
28    sub[1].reimbs={reimbs[2]};
29    }
30 }
```

Fig. 4: `Subscriber` and System classes

4.1 Compilation process

The compilation process is based on a series of model rewritings. This is a powerful tool that allows ODM, not only to abstract instructions from their implementation, but also to conserve the rule paradigm. Practically, the IRL rules life-cycle is completely separated from that of BAL rules. As a consequence, changing the implementation is possible without altering every BR.
When BRs are entered using the BAL, they are first translated into IRL rules by a rewriting procedure. Second, the resulting rule-set is checked and parsed to obtain a rule-set semantic model as Abstract Syntax Tree (AST). At this level, the result may undergo recursive rewritings, on top of which one can plug different APIs. Then, the rule-set AST is compiled while taking into account the chosen algorithm. Again this phase can be parametrized by various plugins according to the algorithm to be used, e.g RETE. The output at this stage is optimized and transformed to obtain the semantic OM. This latter is a powerful meta-model, it can be seen as an extension of the Java meta-model that allows compilation, sources processing and model definition. There is no longer semantic rule-sets here, but instead, an object model that encodes the semantics inside the generated classes and methods. Other operations may appear such as the BAL/IRL mapping and the linkage with outside application via services mechanism. The final result is persisted and jitted into an archive that can be deployed in the desired platform, e.g Java, C# and Script. Note that this chain is executed in pipe-line and the order is controlled by the plugins execution in the chain. Our proposed prototype, called BIS for Bayesian Insight System, can be plugged on top of the rules compilation process as an additional rewriting of the rule-set. The plugging choice is motivated by our desire to take advantage of an existing compilation framework, rather than building such a process from scratch. Additionally, a plugging approach facilitates the conceptual and technical integration in the product architecture. Fig. 5 depicts an overall schema of the compilation process. In particular, a compilation factory is implemented to adapt the probabilistic context to the compilation chain. The IRL-based Rule 1.4 illustrates the results after rewriting the Rule 1.3. In this example, we move from function rewriting to proper call of the probabilistic engine with current arguments.

Rule 1.4: detect invoice anomaly with probability

```
1 rule detectInvoiceAnomaly{
2  when{
3  hp:HealthcareProvider(ProbabilisticEngine.this.
       calculateProbability(this,"price_risk", "high") > 0.8);
4     }
5  then{
6   raiseAlertExceededInvoicePrice(hp);
7     }
8 }
```

When the extended IRL is compiled, annotations serve to extract PRM attributes, CPTs and relations. When the checking is completed, the final model

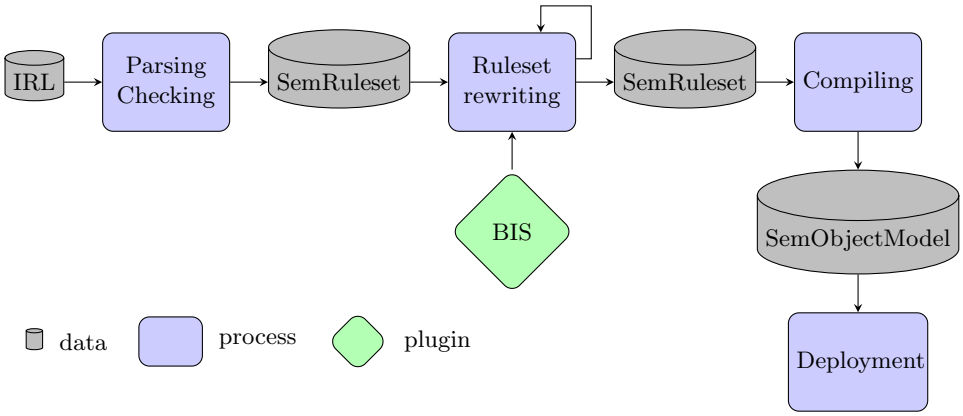

Fig. 5: ODM compiling chain

is written into the PRM text format and processed for inference. We give another way to introduce the probability in rules using the IRL `evaluate` operator. Rule 1.5 evaluates the risk that a `Subscriber` is participating in a fraud.

Rule 1.5: evaluate subscriber risk with probability

```
rule evaluatSubscriberRisk{
 when{
  hp:HealthcareProvider();
  sub: Subscriber() in hp.subs;
  evaluate(prob(sub.risk==true)| hp.risk==true)>.8);
 }
 then{
  alertRiskedSubscriber(sub);
  }
}
```

The vertical bar in the rule stands for "knowing that" and corresponds formally to conditional probability of a RV. In this rule, the RV `sub.risk` is connected to `HealthcareProvider`'s attributes through the reference `subs`. In the expression `prob(sub.risk==true)| hp.risk==true)`, the conditional context is explicitly mentioned, which refers to computing the probability of `sub.risk` given some information on `hp.risk`. However, we want to simplify more the syntax by considering implicitly every fact in the WM. As a consequence, `prob` should be stateful to facilitate the rule writing without bothering oneself with the underlying PRM. Thus, the previous expression is reduced to `prob(sub.risk==true)`, which is implicitly equivalent to `prob(sub.risk==true|WM)`, where WM is simply reduced to `"hp.risk==true"`. Finally, one can similarly manage the introduction of other operators such as `likelihood` and `entropy`.

4.2 Advanced probabilistic rules

At rules level, one is interested in decision-making, to which the notion of *risk* is relevant[6], not in how probabilistic query is performed. So another easy, yet interesting, approach to express probability in the rules, is to parametrize the whole condition by probabilistic activation threshold while allowing the coupling introduced in Section 3. Doing so helps the rule engine agenda to determine which rule should be executed. In practice, we need to introduce a general probability operator that governs rules eligibility by testing if the probability of the corresponding tuple pattern matching equals or exceeds the probabilistic threshold.

Rule 1.6: evaluate subscriber risk with probability

```
rule evaluatSubscriberRisk{
 probability >= .8;
 when{
  hp:HealthcareProvider();
  sub:Subscriber(sub.risk==true|hp.risk==true) in hp.subs;
  }
 then{
  alertRiskedSubscriber(sub);
  }
}
```

The *probability* operator in Rule 1.6 involves all the RVs occurring in the condition part. However, since PRM engines cannot directly deal with such conditions but only with variables, one must specify the rule conditions, which are really participating in computations, and identify the underlying probabilistic variables. This is a challenging task that involves a difficult compilation process. Actually, in this approach, we need to analyze rules, extract information that is relevant to probabilistic inference and avoid non probabilistic variables for instance. Then, one must transform the result into the adequate probabilistic query [7]. Many operators that are present in the compilation process are complex to evaluate and need to be detected. This means introducing new operations in the compilation in order for the probabilistic engine to deal with different conditions including different tests (variable and class conditions), aggregators and generators. Fortunately, both models share some high level operators, e.g, *min, max, for all, exists* aggregators, which can be automatically extracted from rules conditions, thanks to the compilation, and mapped to their PRM counterparts if any. Otherwise they can be used to complete the PRM definition with new attributes.
Now, using this general operator, every logical production rule can be given a probabilistic meaning by considering non probabilistic variables as a Dirac dis-

[6] for instance, the risk of not executing a rule that should be executed (false negative)
[7] in general, one must specify how every language construct is compiled to be processed by this general operator and assure the preservation of queries operational semantic after the rewriting, but this is beyond the scope of this paper

tribution and by imposing *probability* operator to be equal to 1. In this way, we can give probabilistic meaning to Rule 1.1 as showed in Rule 1.7

Rule 1.7: detect invoice anomaly

```
rule detectInvoiceAnomaly{
 probability=1;
 when{
  hp:HealthcareProvider();
  sub:Subscriber() in hp.subs();
  exists Reimbursement(price=='high') in sub.reimbs();
 }
 then{
  raiseAlertExceededInvoicePrice(hp);
 }
}
```

4.3 A loosely coupling-based execution

The declarative aspect of the rules facilitates efficient instances generation for the PRM system. At run-time, a hundred of RVs with complex relations can be easily obtained just by means of instanciation; this is an advantage over the classical BNs approach. In addition to the compilation API discussed previously, BIS is also endowed with an execution API. Both insure different services communicating following the schema shown in Fig. 6. This allows for a coupling between both BRs and probabilistic engines, which are implemented as services. Actually, our framework is not restricted to one implementation, but is open to any other probabilistic engine, which can be seen as a plugged service implementation. For instance, the current work is using aGrUM that can deal with

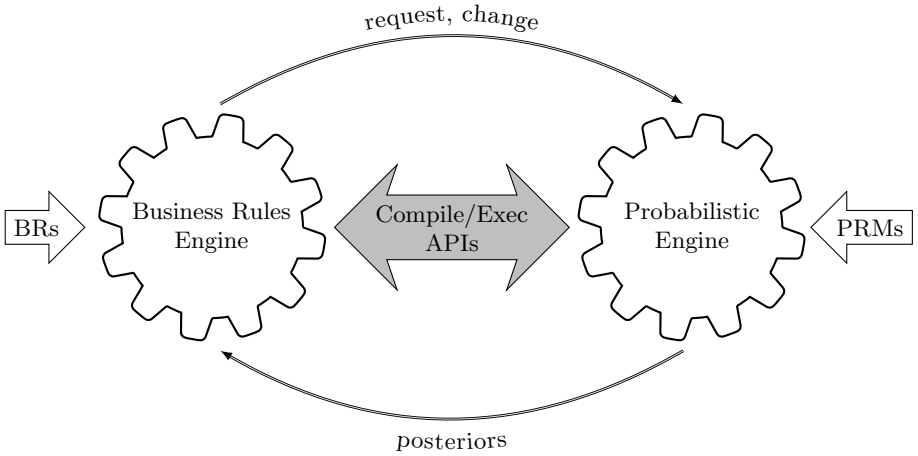

Fig. 6: BIS coupling

PRMs. We have also tested JSmile[8] as a probabilistic engine, however, we were limited by the lack of relations and object concepts in such a framework. Recall that the compilation part performs a rewriting from the rules semantic model, which encompasses probabilistic data, to run-time functions, which actually call the probabilistic engine. In our case the PRM is generated by XOM compilation and read by probabilistic engine. It is also possible to read both models from external files. Furthermore, our architecture allows for a good inter-operability between both engines. On the one hand, rules execution can change the state of the WM and consequently the RVs in the PRM system by posting evidence, adding, removing new instances or setting new relations. For instance, the action part of Rule 1.8, update the WM by adapting the `risk` attribute. Through additional process of rewriting and compilation, one can even discover particular variables to post soft evidence on their corresponding probabilistic ones [9].

Rule 1.8: detect invoice anomaly with probability

```
rule detectInvoiceAnomaly{
 when{
  hp:HealthcareProvider( prob(type_risk==high)>.8) );
  sub: Subscriber() in hp.subs;
 }
 then{
  update sub{risk=true;}
     }
}
```

On the other hand, when the rules trigger the inference process, the probabilistic engine computes the needed probability for the query and may also notify the WM to update some attributes for rules reevaluation.
In Fig. 7 engines are related via an observation mechanism and both are notified,

[8] see https://dslpitt.org/genie/wiki/JSMILE_and_Smile.NET for more details
[9] constrains on probability distribution

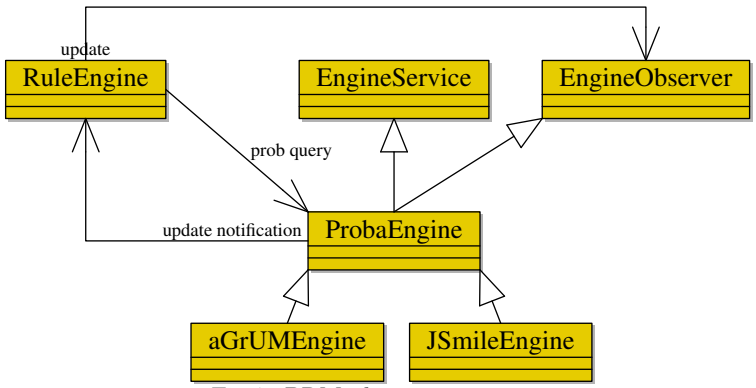

Fig. 7: PRM plugin as a service

through the observer, when any change occurs in the WM. That is, each time an incremental change, such as object or relation insertion/retraction occurs in the WM, the underlying PRM is notified for the update, e.g, reference slot change. Both probabilistic and rules engine perform inference in a lazy way. The former records every incremental WM update until the next rule query evaluation. Then it accordingly updates the PRM system to answer the probabilistic query. The latter seeks to minimize evaluations in the RETE network.

5 Conclusion

This paper introduced an effective approach to integrate probabilistic reasoning into modern BRMS. The solution we proposed couples BRMS with PRMs. We highlighted the natural mapping between both paradigms and gave an operational method to assure it. Finally we proposed a general architecture of the coupling platform prototype. The technical contribution is implemented as an embedded prototype in the ODM product.

Our work opens many other research perspectives. In particular, due to the dynamic aspect of the WM, a probabilistic inference algorithm should be developed to insure the PRM inference adaptability. For this purpose, we are currently working on an adaptation of the incremental junction tree inference algorithm, which is proposed in [2]. Moreover, as soon as we deal with aggregators/generators in the rules, e.g, `from,in`, it becomes necessary to automatically generate their PRM counterpart as and when we compile the rules. This immediately opens the issue of PRMs non-supported operators and the idea of extending this model. Another difficult compilation aspect, yet more interesting, is the use of several operators within the same rule and how this impacts the PRM construction.

After each WM update, the RETE algorithm tries to optimize the expression re-evaluations, while taking into account the previous state. It would be interesting to develop a two-sided optimization that supervises PRM and WM updates, e.g, take into consideration the PRM variables independence. Or better yet, to try to optimize updates propagation as part of a tight coupling but to the cost of algorithmic complexity and finding a trade-off between implementation and performance. Finally, this work also opens doors to introduce uncertain reasoning in rules temporal expressions to process complex events over uncertain data or events.

Acknowledgments: This work was partially supported by IBM France Lab/ANRT CIFRE under the grant #421/2014. The authors would like to thank Christian De Sainte Marie for useful discussions and insights.

References

1. Agli, H., Bonnard, P., Wuillemin, P., Gonzales, C.: Uncertain reasoning for business rules. In: Proceedings of the 8th International Web Rule Symposium Doctoral Consortium (2014)

2. Agli, H., Bonnard, P., Wuillemin, P., Gonzales, C.: Incremental junction tree inference. In: Proceedings of the 16th Information Processing and Management of Uncertainty in Knowledge-Based Systems International Conference (2016), to appear
3. Berstel-Da Silva, B.: Verification of Business Rules Programs. Springer (2014)
4. Bobek, S., Nalepa, G.J.: Compact representation of conditional probability for rule-based mobile context-aware systems. In: Proceedings of the 9th International Symposium, RuleML. pp. 83–96 (2015)
5. Buchanan, B.G., Shortliffe, E.H.: Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project. Addison-Wesley (1984)
6. De Raedt, L., Kimmig, A.: Probabilistic (logic) programming concepts. Machine Learning 100(1), 5–47 (2015)
7. Elkan, C.: The paradoxical success of fuzzy logic. In: IEEE Expert. pp. 698–703 (1993)
8. Forgy, C.L.: RETE: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence (1982)
9. Graham, I.: Business Rules Management and Service Oriented Architecture: A Pattern Language (2006)
10. Hart, P.E., Duda, R.O., Einaudi, M.T.: PROSPECTOR—a computer-based consultation system for mineral exploration. Journal of the International Association for Mathematical Geology 10(5), 589–610 (1977)
11. Heckerman, D.E., Shortliffe, E.H.: From certainty factors to belief networks. Artificial Intelligence in Medicine 4(1), 35–52 (1992)
12. Koller, D., Pfeffer, A.: Probabilistic frame-based systems. In: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI). pp. 580–587 (1998)
13. Koller, D., Pfeffer, A.: Object-oriented Bayesian networks. In: Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI). pp. 302–313 (1997)
14. Korver, M., Lucas, P.J.F.: Converting a rule-based expert system into a belief network. Medical Informatics 18, 219–241 (1993)
15. Mahoney, S.M., Laskey, K.B.: Network engineering for complex belief networks. In: Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence (UAI). pp. 389–396 (1996)
16. Ng, K.C., Abramson, B.: Uncertainty management in expert systems. IEEE Expert: Intelligent Systems and Their Applications 5(2), 29–48 (Apr 1990)
17. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
18. Pfeffer, A.J.: Probabilistic Reasoning for Complex Systems. Ph.D. thesis, Stanford University (2000)
19. Torti, L., Gonzales, C., Wuillemin, P.H.: Speeding-up structured probabilistic inference using pattern mining. International Journal of Approximate Reasoning 54(7), 900–918 (2013)
20. Wuillemin, P.H., Torti, L.: Structured probabilistic inference. International Journal of Approximate Reasoning (2012)
21. Zadeh, L.A.: Fuzzy sets. Information and Control 8(3), 338–353 (1965)