Encapsulation-Driven Approach to Interchange of Knowledge Base Structure

Krzysztof Kaczor and Grzegorz J. Nalepa

Abstract—Rules provide an efficient way of decision modeling. This is why they are commonly used in knowledge based decision support systems as knowledge representation method. During the years, many different rule representations providing different syntax, semantics and expressive power have been developed. Therefore, the problem of efficient rule interoperability became very challenging. Differences between existing rule representations can also be noticed in the context of knowledge base structure and cannot be omitted by interoperability methods. This paper discusses how the knowledge base structure cannot be omitted while modeling decision process can be expressed in other representations with the help of knowledge encapsulation mechanism.

Index Terms—Semantic rule interoperability, knowledge structure, decision support systems, rule-based systems.

I. INTRODUCTION

Rules are one of the most successful methods for knowledge representation within Artificial Intelligence (AI). They provide an efficient, intuitive and declarative way for knowledge encoding. This is why rules are applied within many different areas like Semantic Web (SW), Rule-Base Systems (RBS), Decision Support Systems (DSS), Complex Event Processing (CEP) [1], or Business Processes (BPs) [2]. The diversity of rules application areas caused development of diverse rules representations. These representations can differ in many aspects e.g. they can use different types of rules, they can be based on different formalisms or they can be just programming solutions. From this diversity stems a problem of efficient rule interchange related to the way how to model knowledge expressed in one representation within another one. The solution of this problem is not trivial because efficient interoperability method must take several issues into account. In [3], we distinguished three levels of abstraction involving these issues: 1) Syntax level where the interchange method translates syntax of the rule language provided by the given representations, 2) Semantic level on which the interchange process must preserve semantics of rules and 3) Structure level where the structure of the knowledge base is taken into consideration.

The first two levels seem to be obvious in the context of interchange. Nevertheless, the structure level plays also a significant role because it has an impact on the inference process. Due to this fact, depending on the knowledge structure, the same inference algorithm may draw different conclusions for a certain knowledge base that contains semantically equivalent rules. This is why the aspect of the rule base structure cannot be omitted while modeling decision process.

Majority of the existing representations provide encapsulation mechanisms that allow for modeling structured rule bases with the help of so-called modules. Nevertheless, differences involving expressive power of the representations have also impact on these mechanisms which supports different types of constructs having impact on the structure. The original contribution of this paper is to propose a way of reflecting selected types of rule base structures with the help of encapsulation mechanisms. This is an important issue especially in the case when a given rule representation is not expressive enough to model a certain rule base structure that is specific for another representation.

The paper is organized as follows: First, the scope of this work is described in Section II. Then, the motivation for this research is presented in Section III. Section IV provides a formal definition of the knowledge base structure in terms of modules. It additionally gives an algorithm that shows how a certain structure of a rule base should be processed by an inference engine. Section V provides an analysis of the different methods that are used to model a given structure of a rule base and discusses their advantages and drawbacks. Section VI is the main part of this paper and provides a discussion how to model rule base structure with the help of modules. Issues related to evaluation of proposed approach are in Section VII. In Section VIII aspects related to future work are described.

II. SCOPE OF THE WORK

The problem of rule interoperability is complex. Even the well-known approaches to this problem are unable to cope with it entirely. Thus, one can distinguish two types of such approaches: The first type provides framework combining different rule representations on one hand, but on the other, due to the diversity of them, it is usually divided into the so-called dialects specialized for single representations. As examples of such approaches one can consider methods like Rule Interchange Framework (RIF) [4] or Rule Markup Language (RuleML) [5]. The second type includes methods that are specialized for selected representations e.g. Production Rule Representation (PRR) [6] or Common Logic (CL) [7]. Nevertheless, none of the existing rule interoperability methods do not take rule base structure into account. This makes them useless in the context of rule bases having structure.
The work, described in this paper, can be classified to the second type, because it is focused on forward chaining systems that use production rules. In this area, there are several commonly known tools that can be taken into account. These are: CLIPS [8], JESS [9], DROOLS [10], OPENRULES3 and ILOG2. The remaining part of the paper, which involves issues related to business rules, should be considered within the context of such rule languages.

III. MOTIVATION

The motivation for this work is a part of motivation for research concerning efficient rule interoperability method. According to the previous results presented in [11] and [12], knowledge structure must be taken by interchange method into account. Moreover, majority of rule-based tools considered in this research allow for modeling structured rule bases. The most basic element used for this purpose is called module2. Each module is a set of rules. During the inference, modules are focused in a way that only one module can be focused at once. An inference algorithm searches for rules that have satisfied their conditional parts. Having a set of such rules, only rules that belong to the focused module can be scheduled for execution (knowledge encapsulation). In this way, the module mechanism determines the amount of knowledge that is available in a given point of inference and order of rule execution.

Apart from modules, rule languages may also provide other constructs that are considered to have an impact on a knowledge base structure. We call such constructs partitioning features because usually they are provided as rule properties that have an impact on the order of rule evaluation within single module. The main problem of this research lies in the fact that these features are often representation-specific constructs and are not supported by other representations. Thus, in the context of rule interoperability, this paper presents how to express decision process structure provided by partitioning features with the help of modules.

IV. DEFINITION AND INTERPRETATION OF RULE BASE STRUCTURE

This paper focuses on the three most important partitioning features: no-loop, lock-on-active and activation-group. The no-loop property prevents a certain rule to be reactivated due to the changes in working memory made by this rule. In turn, lock-on-active can be considered as an extended version of no-loop. It disallows all rules having this property, to be reactivated until the focus is removed from the module. The last property is called activation-group. If a rule provides this property, then it cannot be activated until there is other pending activation of rule having the same value of this property. For more precise clarification how these features work, the formal definition of module and internal module structure is provided in the Subsection IV.A and Subsection IV.B. Furthermore, the generalized algorithm-like definition of inference process, taking these features into account, is provided in Subsection IV.C.

A. Modules

The set containing all rules can be divided into several non-empty pairwise disjoint subsets that are called modules:

\[ \Gamma = M_1 \cup M_2 \cup \cdots \cup M_n \]

where 1, 2, ..., n are module identifiers. According to this definition, a single module is a set of rules. In turn, \( M_i \) belongs to the ordered set of all modules \((\mathcal{M}, \prec_M)\), where the \( \prec_M \) relation defines the order of the modules. Using this relation the min\( _\mathcal{M} \) function that determine the minimal element of \((\mathcal{M}, \prec_M)\) can be defined:

\[ \min_\mathcal{M}((\mathcal{M}, \prec_M)) = M \iff \bigwedge_{M_i \in \mathcal{M} \setminus M \prec_M} (M_1 \prec_M M_i) \]

The min\( _\mathcal{M} \) function is important in the context of modules evaluation order. Based on this function, the next\( _\mathcal{M} \) function, that in terms of the provided ordering relation determines the next module with respect to a certain module \( M_i \), can be defined in the following way:

\[ \text{next}_\mathcal{M}(M_i) = M_j \iff M_j = \min_\mathcal{M}((\mathcal{M}\setminus\{M_1, M_2, \cdots, M_k\}, \prec_M) \]

where \( \{M_1, M_2, \cdots, M_k, M_i\} \) is the set of all modules belonging to \((\mathcal{M}, \prec_M)\) such that \( \bigwedge_{M_i \in \{M_1, M_2, \cdots, M_k\}} M \prec_M M_i \).

B. Structure of Single Module

In order to provide support for other features of rules like activation-group or lock-on-active, each single module introduces its own structure. Each module is divided into several submodules that correspond to these features of rules. This is why, a structure of a single module \( M_i \) is defined as follows:

\[ M_i = M_i^o \cup M_1^i \cup M_1^{i\oplus 1} \cup M_1^{i\oplus 2} \cup \cdots \cup M_1^{i\oplus n} \]

where:

- \( M_i^o \) is an ordinary submodule and contains only ordinary rules i.e. rules that have not defined any of lock-on-active and activation-group features.
- \( M_1^i \) is an lock-on-active submodule and contains rules with lock-on-active feature. There can be only one such submodule within single module.
- \( M_1^{i\oplus k} \) is called xor submodule and contains rules with activation-group feature. In contrast to lockon-active submodule, there can be more than one xor submodule within single module \( M_i \).
- \( n \) is the number of xor submodules.
- It is assumed that an ordinary submodule is disjoint with all the remaining submodules, while all the submodules, besides ordinary one, can contain the same rules i.e. they are allowed to have a non-empty intersection.

The next part of this section shows how the modules and submodules should be interpreted by inference algorithm.
C. Interpretation of Rule Base Structure

Modules provide an additional level of abstraction for the knowledge stored within the knowledge base. This level defines structure of knowledge base that cannot be omitted in the context of efficient interoperability method. All of the considered rule languages use some variation of modules. What is more, they use forward chaining inference mode for processing production rules. This is why, the generalized algorithm-like definition of knowledge base structure may be similar to forward chaining inference algorithm.

First of all, the list of symbols, that have not been defined as far, is provided in order to make this definition more clear:

- $M_c$ denotes the current module i.e. the module that has focus.
- $r$ stands for a certain rule instance. In general, the definition of rule is not relevant within this paper. Nevertheless, one assumption concerning rules is provided: each rule takes one argument being a set of rules that can be activated, and as the result, it updates this set by adding or removing some elements. Thanks to this assumption, each rule can provide inference control by moving focus between modules as well as an interpretation of no-loop property can be defined.
- $\Gamma_S$ is the set of rules that can be activated (added to conflict set).
- $\Gamma_I$ is the set of all rule instances that were not executed. This set is usually created by pattern matching algorithms like RETE [13], TREAT [14] or GATOR [15], however it can also be created using more naive algorithms.
- $CS, <_{CS}$ refers to a conflict set that contains instances of rules that can be potentially executed.
- $<_{CS}$ is a rule ordering relation provided by a conflict set (conflict set resolution strategy). Using this relation the $\min_{CS}$ function can be defined in the similar way as the $\min_M$ function. This relation determines the order of rule execution.

Having all definitions, the assumed algorithm of modules interpretation can be defined as follows:

1. $M_c := \min_M(M_c, <_M)$;
2. $(M_c, <_M) := (M_c, <_M) \cup \Gamma_I$;
3. $\Gamma_S := M_c$;
4. Create set $(\Gamma_I, <_{CS})$;
5. If $(\Gamma_I, <_{CS}) \neq \emptyset$ then iterate through all elements of $(\Gamma_I, <_{CS})$ in the order determined by $<_{CS}$:
   - Select next rule instance $r$ in the following way:
     - $r = \min_{CS}(\Gamma_I, <_{CS})$;
     - If $r \notin \Gamma_I$ then go to 5a;
     - If $r \in \Gamma_I$ then go to 5a;
     - If $r \in \Gamma_I$ then perform step 2 and later go to 1;
     - If $M_c \in \Gamma_I$ then $\Gamma_S := M_c \setminus M_c$ and go to 4;
     - If $M_c \notin \Gamma_I$ then find module $M_j$ such that $M_c \subset M_j$ and $M_c := M_j$ and go to 3;
6. If $(CS, <_{CS}) \neq \emptyset$ then select rule instance $r := \min_{CS}(CS, <_{CS})$;
   - $(CS, <_{CS}) := (CS \cup \{r\}, <_{CS})$;
   - $(\Gamma_I, <_{CS}) := (\Gamma_I \setminus \{r\}, <_{CS})$;
   - If $(\Gamma_I, <_{CS}) \neq \emptyset$ then go to 5a;
    
7. If $(CS, <_{CS}) \neq \emptyset$ then perform step 2 and later go to 1;
8. End;

Within this algorithm-like definition, several important points require additional explanation:

- The main purpose of the $\Gamma_S$ set is to make this definition independent from the type of pattern-matching algorithm. Thanks to that, a step of pattern-matching can be performed by even a naive algorithm that always finds all the rules having satisfied their conditional parts, but only the selected rules are allowed to be added to a conflict set.
- The step 5) uses the $<_{CS}$ relation in order to assure that a certain rule belonging to some xor submodule, is added to a conflict set before the other rules from this submodule that have lower priority.
- Rules providing activation-group property cannot be added to a conflict set when other rule with this property is already there. Then such rule instance is removed (see step 5).
- The step allows for executing only these rules that belong to the current module $M_c$. Execution of other rules is canceled.
- The inference mechanism provides information concerning current module into rule as their argument (see step 5). It is worth to notice that the lock-on-active submodule is excluded from this information. This is why, according to the nature of this submodule, these rules can be activated only at the moment when their module becomes a current module.
- The step considers possible changes in execution control made by rules. Each rule can change a current module – the case described by step 5, or they can affect the $\Gamma_S$ set – e.g. rules having no-loop feature.

The provided definition is consistent with the inference engine implemented in DROOLS that provides a wide support for different rule features and rule execution control that affect knowledge structure. Furthermore, this definition can also be used to describe modules interpretation in other representations.

V. EXISTING MODELING TECHNIQUES

The partitioning features considered in this paper are mainly used to control rule activations caused by the rule itself or by other rules. The main goal of controlling rule activations is to prevent rule engine from going into infinite loops during inference. An inference engine may be looped even in case of trivial rules. As an example consider the following discount rule, where its reactivation is caused by the modification made in the conclusion part:

```
rule "Apply more than $100 discount"
    when
    $o: Order(value > 100)
    then
    modify($o)
```

Many of the existing rule languages do not support any of the discussed features. Thus, in order to avoid such undesirable situations, a number of alternative solutions are provided.

First of the considered solutions requires modification of fact model in order to provide the so-called control fact fields. In this approach, each fact type that can cause infinite rule reactivations is enriched by a dedicated and rule-specific field that stores information concerning what kind of modification were applied to this fact instance. What is more, each potentially dangerous rule has to provide an additional condition that will check this field in order to verify if the modification have already been applied or not. Using this approach, the provided discount rule can be defined as follows:

```java
   setDiscount(o.getDiscount()+0.1),
   addDiscountReason("Order above $100")
}
End
```

The main disadvantage of this approach is related to maintenance of such rules and facts. This is due to the case that fact model, besides business-related fields, must provide rule specific ones. What is more, there is no precise distinction between these two types of fields what make the business logic unclear.

In order to improve the maintainability of the business-related facts, the other approach called control facts [16], [17] can be applied. This approach allows for embedding execution information into the domain knowledge by adding new types of facts. An instance of such fact type is intended to represent the execution of a specific rule for a specific fact set. It can be noticed that such facts play a similar role as the control fields. Nevertheless, in comparison to control fields, they improve flexibility of storing control information and makes the business logic more clear. As an example one can consider the provided discount rule that uses DiscountApplied control fact:

```java
   rule "Apply more than $100 discount"
when
   $o: Order(value > 100, 
   discountReasons not contains "Order above $100")
then
   modify($o){
      setDiscount($o.getDiscount()+0.1),
      addDiscountReason("Order above $100")
   }
End
```

The remaining part of this section, elaborates how to model considered rule partitioning features with the help of knowledge encapsulation determined by modules. In order to make the proposed approach clearer, demonstrative diagrams that use this notation are provided.

The no-loop feature can be considered as the simplest one. If a certain module contains rules providing this feature, then it can be considered as module having internal structure that is depicted in Fig. 2. This structure groups rules having the same priority into single submodules in a way where each rule providing no-loop feature must be separated from any other rules by placing it into separate module. Rules without this feature and having the same priority can be placed into one module. It is assumed that default order of submodules

VI. MODULE-BASED MODELING OF RULE BASE STRUCTURE

The original contribution of this paper is to propose an approach to represent rule base structure with the help of modularization mechanism that is provided by majority of production rule languages. The structure of the decision process is determined by the modules mechanisms as well as other features that are provided by single rules. In comparison to the previously presented solutions, this approach tries to avoid persistent mixing of business logic and control information. Moreover, using this approach, the considered within this paper rule features can be modeled in more efficient way.

The most basic decision process structure is determined by modules. During inference process modules are evaluated in a certain order that usually is defined in advance. We call it default order and it is specified by the $\prec_M$ ordering relation. However, this order can be modified by rules that may change the currently evaluated module (see Section IV.C, point ).

Modules as well as order of their evaluation can be easily presented in a graphical way. The Fig. 1 shows a simple sequence of modules. Each connections between them define the possible order of their evaluation. The connections represented with the dashed line show the default order of evaluation that is used when there is no more rules for execution in the module. In turn, connections displayed with the help of solid lines correspond to alternative (with respect to default order) evaluation orders that can be enforced by specific rules.

![Fig. 1. Sequence of modules evaluation.](image-url)

The no-loop feature can be considered as the simplest one. If a certain module contains rules providing this feature, then it can be considered as module having internal structure that is depicted in Fig. 2. This structure groups rules having the same priority into single submodules in a way where each rule providing no-loop feature must be separated from any other rules by placing it into separate module. Rules without this feature and having the same priority can be placed into one module. It is assumed that default order of submodules
evaluation is consistent with rule priorities ordered in descending manner. Thus, according to the Fig. 2 the following relation takes place: \( a \geq b \geq c \geq \cdots \geq n \).

![Fig. 2. Structure determined by the no-loop feature.](image)

In order to assure correct rules evaluation order, each rule must provide an appropriate control statement within their conclusion part. One can distinguish two major contexts of rule execution. First is related to execution of rule without no-loop feature. If the priority of such rule is less than \( a \), then such a rule must move focus to module having highest priority rule(s). If this is the highest priority rule, then such rule does not have to provide any control statements. The second context of rule execution involves rules with no-loop feature. In order to avoid infinite loop, each no-loop rule must change a current module. In the case of the highest priority rule, focus is moved to the next module. In the remaining cases, focus must be moved to the first module.

![Fig. 3. Structure determined by the activation-group feature.](image)

The structure corresponding to activation-group feature is similar to structure providing by no-loop. This is why, in order to make discussion concerning this feature more clear, issues related to rule priorities are omitted. The activation-group feature allows rules to be executed exclusively. It is assumed that there can be more than one activation group within one module. Thus, rules that belong to the same activation group are separated from other rules. This determines the internal structure of the module that can be depicted as in Fig. 3. Because all the rules have the same priority, any order of their activation and execution should lead to the same result. This is why, this approach does not define the default order of the A-G submodules. Nevertheless, each rule with activation-group feature must provide appropriate control statement in the conclusion part that will move the focus to the next module in order to prevent from activating other rules belonging to this submodule (see Section IV.C, point \( \star \)). After evaluation of all the A-G submodules, rules without activation-group feature can be evaluated. Evaluation of the submodules depicted in the Fig. 3 is in the form of loop. This loop continues until no rules are executed during single iteration. In order to ensure the correct number of loop iterations, introduction of control fact and rule, that checks for its existence, may be required. Nevertheless, in comparison to control facts approach discussed in Section V, such control fact does not store any business information and is separated from the business logic. This is why, this fact is considered as temporal not as persistent.

The lock-on-active feature allows rules to be evaluated only against facts that are available at the moment of receiving focus by the module. Any changes of facts that are made later cannot activate such rules again. In the proposed approach rules providing lock-on-active feature are separated from other rules by placing them into different submodule (submodule L-O-A in Fig. 4). This separation prevents from reactivation of such rules by the remaining ones. However, the most important issue related to this feature involves mutual reactivations of the rules providing this feature. In order to avoid this problem, the mutual dependencies between rules must be eliminated. For this purpose, this approach uses control facts (see Section V) in the following way: Each lock-on-active rule can be evaluated and instantiated for any combination of business facts. However, instead of direct modifications of the facts, rule produces control facts that store information about intended modifications. In this way, rule reactivations caused by the mutual dependencies are not possible. On one hand such solution causes intermixing of business and control information, but on the other, this control information is not persistent. It is removed after the focus is moved from the L-O-A submodule to submodule containing remaining rules. Just after the second submodule receives focus, dedicated rules, having highest priority, merge information stored within control facts into business facts and removes the control ones. In this way, the information provided by the lock-on-active rules are stored within business facts and thus, the further inference does not need to have any information concerning control facts model and is independent from their instances.

![Fig. 4. Structure determined by the lock-on-active feature.](image)

This section presented how the selected partitioning features can be modeled with the help of modules. The remaining sections describe evaluation of this approach and the nearest future works.

VII. EVALUATION OF THE APPROACH

The presented approach was tested on the two use cases called PLOC\(^4\) and UserV [18]. These cases consist of 30 and 80 rules, respectively and require of using considered rule features. They are implemented in two ways: with the help of partitioning features (DROOLS-based implementation) and using the proposed approach (CLIPS-based implementation). Execution of them gives the same result for different inputs and, what is more, the rule execution order can be considered as the same in terms of decision process structure:

Execution of UserV example modeled in DROOLS:

```
automobile-eligibility-potential-theft:::FT3-high3:
automobile-eligibility-potential-occupant-injury::
POIR-moderate:
automobile-eligibility-calculation::provisional2:
driver-eligibility-driver-certification::
has-not-certificate:
hekate_case_ploc
```

driver-eligibility-calculation::eligible3:
eligibility-scoring::create-initial-sum-of-scoring :
eligibility-scoring::sum-and-display-scoring:
eligibility-scoring::sum-and-display-scoring:
eligibility-scoring::sum-and-display-scoring:
eligibility-scoring::summary:
eligibility-scoring::decision-eligible1:
car-base-premiums::base-premium-car-compact:
car-premiums::create-initial-sum-of-car-premiums:
car-premiums::sum-and-display-car-premiums:
car-premiums::sum-and-display-car-premiums:
driver-premiums::create-initial-sum-of-driver-premiums:
driver-premiums::sum-and-display-driver-premiums:
driver-premiums::sum-and-display-driver-premiums:
car-discounts::create-initial-sum-of-car-discounts :
car-discounts::sum-and-display-car-discounts:
car-discounts::sum-and-display-car-discounts:
market-segment-discounts::
create-initial-sum-of-market-segment-discounts:
market-segment-discounts::
market-segment-discounts::
sum-and-display-market-segment-discounts:

Final premium for client KKR is: 250$

Execution of UserV example modeled in CLIPS:

automobile-eligibility-potential-theft::PTR-high3:
automobile-eligibility-potential-occupant-injury::
POIR-moderate:
automobile-eligibility-calculation::provisional2:
driver-eligibility-driver-certification::
has-not-certificate:
driver-eligibility-calculation::eligible3:
eligibility-scoring::create-initial-sum-of-scoring :
eligibility-scoring::sum-and-display-scoring:
eligibility-scoring::sum-and-display-scoring:
eligibility-scoring::sum-and-display-scoring:
eligibility-scoring::summary:
eligibility-scoring::decision-eligible1:
car-base-premiums::base-premium-car-compact:
car-premiums::create-initial-sum-of-car-premiums:
car-premiums::sum-and-display-car-premiums:
car-premiums::sum-and-display-car-premiums:
driver-premiums::create-initial-sum-of-driver-premiums:
driver-premiums::sum-and-display-driver-premiums:
driver-premiums::sum-and-display-driver-premiums:
car-discounts::create-initial-sum-of-car-discounts :
car-discounts::sum-and-display-car-discounts:
car-discounts::sum-and-display-car-discounts:
market-segment-discounts::
create-initial-sum-of-market-segment-discounts:
market-segment-discounts::
market-segment-discounts::
sum-and-display-market-segment-discounts:

Final premium for client KKR is: 250$

Rule base of this example is divided into several modules.

Rule base of this example is divided into several modules.
Due to the limited size of the paper only one module is
presented in order to show how the proposed approach can be
applied in practice. The module
driver-eligibility-driver-record defined
with the help of DROOLS contains three rules providing both
lock-on-active and activation-group
partitioning features:

rule "driver-eligibility-driver-records::high-risk-driver1"
agenda-group "driver-eligibility-driver-records"
activation-group
"driver-eligibility-driver-records"
lock-on-active true
when
Client ($Name : Name, DuI == true)
$ClientFacts : ClientFacts(Name == $Name)
then
modify($ClientFacts) {setIsHRD(true)}
System.out.println("driver-eligibility-driver-records::
high-risk-driver1:"")
end

rule "driver-eligibility-driver-records::high-risk-driver2"
agenda-group "driver-eligibility-driver-records"
activation-group
"driver-eligibility-driver-records"
lock-on-active true
when
Client ($Name : Name, NoA > 2)
$ClientFacts : ClientFacts(Name == $Name)
then
modify($ClientFacts) {setIsHRD(true)}
System.out.println("driver-eligibility-driver-records::
high-risk-driver2:"")
end

rule "driver-eligibility-driver-records::high-risk-driver3"
agenda-group "driver-eligibility-driver-records"
activation-group
"driver-eligibility-driver-records"
lock-on-active true
when
Client ($Name : Name, NoMV > 3)
$ClientFacts : ClientFacts(Name == $Name)
then
modify($ClientFacts) {setIsHRD(true)}
System.out.println("driver-eligibility-driver-records::
high-risk-driver3:"")
end

Considering proposed approach, it is important to notice
that all the rules within this module provide
lock-on-active feature. Thus, according to Fig. 4, the
module named other rules is empty and hence it can be
omitted in during the translation. Furthermore, each rule
defines the same activation-group. Therefore, each
corresponding rule in CLIPS must provide
(pop-focus) control statement that moves the focus to another module.
Because the other rules (see Fig. 3) module is also empty, it
can be omitted too. Taking translation of both features, the
CLIPS rules can be defined as follows:

{deffact
driver-eligibility-driver-records::high-risk-drive
r1 (Client (Name ?Name) (DuI true))
?ClientFacts <- (ClientFacts (Name ?Name))
->
(modify ?ClientFacts (IsHRD true))
(printout t "driver-eligibility-driver-records::
high-risk-driver1:" crlf)
concerning how the combinations of the rule language features affect structure of single module is an important issue of future works.

The approach presented in this paper is a part of research that focuses on providing an efficient rule interoperability method that takes semantics and structure of the knowledge base into account. This method is intended to be supported by tools in order to make it more practical. The future work also includes an implementation of a proof-of-concept tool that is expected to provide support for interchanging knowledge base structure using this approach.

REFERENCES


Krzysztof Kaczor was born in Zakopane, Poland in 1984. He received his MSc degree in the field of computer science in 2008 from AGH University of Science and Technology in Krakow, Poland. In his master thesis he is focusing on design and implementation of a unified rule base editor allowing for visual modeling of the XTT2 rule bases. Since 2007 he has been actively involved in the number of research projects, where his work was mainly related with the visual and formal knowledge representation and processing (Hekate, Rebit, Bimloq). Since 2008, he held a position of a research assistant at the AGH UST in Krakow, Poland, the Department of Automatics. Currently, his work and PhD dissertation concerns the development of unified and formalized model for rule representation allowing for semantic business rule interoperability.

MSc Kaczor is a founder member of Polish Artificial Intelligence Society.