

CLIPS Representation of Ontology Classes in an Ontology-Driven Information System Builder – Part 2

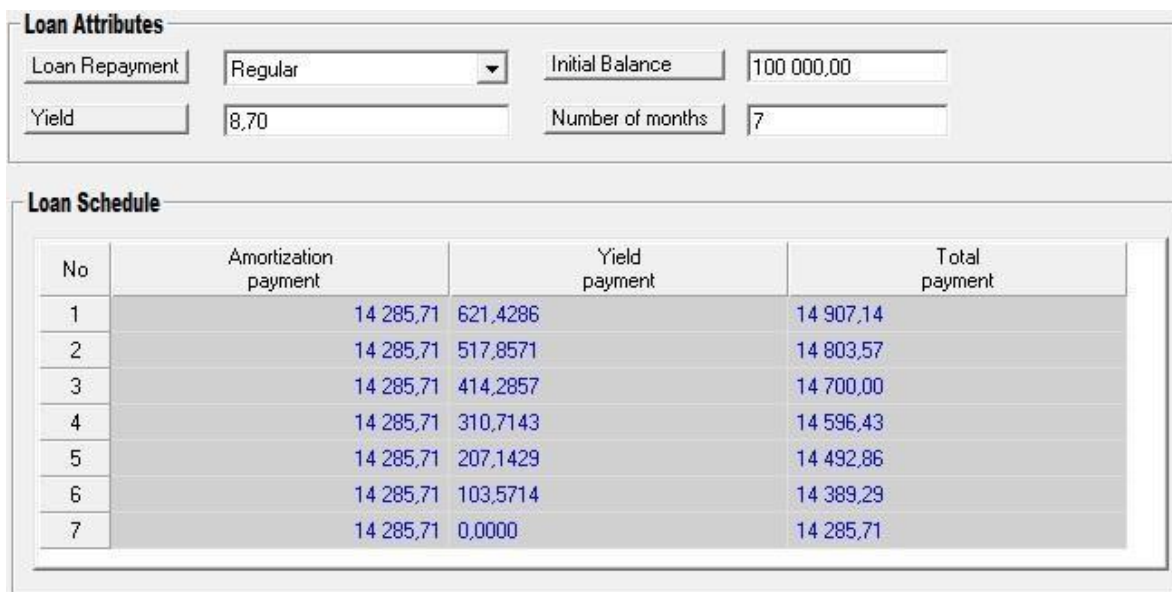
Samuil Nikolov

Abstract: The paper describes the structure of a CLIPS program representing an ontology class in an ontology-driven information system builder. The program has a specific format which is presented in the paper. A set of programs can be used to quickly and easily create an ontology-driven information system when loaded in a C++ core which interoperates with an instance of a CLIPS interpreter. An example bank loan class expressed as a CLIPS program is presented in the second part of the publication.

Keywords: Ontology-Driven Information Systems, Knowledge Representation, Expert systems

2. Example for building a CLIPS program for calculating bank loans

The structure of an example program will be presented that defines the UI and calculates several parameters of a simple loan. Each of the three program parts introduced in the first part of the publication will be explained separately following the same subsection naming. The user interface(UI) generated by the information system builder when loading the example program is shown on Fig. 2.



Loan Attributes			
Loan Repayment	Regular	Initial Balance	100 000,00
Yield	8,70	Number of months	7

Loan Schedule			
No	Amortization payment	Yield payment	Total payment
1	14 285,71	621,4286	14 907,14
2	14 285,71	517,8571	14 803,57
3	14 285,71	414,2857	14 700,00
4	14 285,71	310,7143	14 596,43
5	14 285,71	207,1429	14 492,86
6	14 285,71	103,5714	14 389,29
7	14 285,71	0,0000	14 285,71

Figure 2 User Interface generated when loading the example CLIPS program.

2.1 Class attributes description

The represented ontology class is simplified for illustration purposes and has just four attributes – loan repayment type, loan initial balance, the paid yield and the loan term in months. The loan repayment type attribute is of selectable text type. As such, it requires a list of values, which is defined at the beginning of the program with the following fact declaration:

```
(list_LOAN_TYPE "Bullet" "Regular" "Annuity")
```

The attribute definition is the same as in (2), but the template "list" slot has the name of the above fact as default value. Also, the template is named LOAN_TYPE:

```
(deftemplate LOAN_TYPE
  (slot type (type STRING) (default "ComboText"))
  (slot str_value (type STRING) (default "")) (8)
  (slot value (type INTEGER) (default 1))
  (slot list (type STRING) (default "list_LOAN_TYPE")))
```

Following are the other three attributes of the class, which are defined as floating point numbers. They have the same form as (1), but the slot “type” has default value “Number” and the templates’ names are ID_BALANCE, ID_YIELD and ID_MONTHS for each ontology class attribute:

```
(deftemplate ID_BALANCE
  (slot type (type STRING) (default "Number"))
  (slot str_value (type STRING) (default ""))
  (slot value (type FLOAT) (default 0.000000)))
(deftemplate ID_YIELD
  (slot type (type STRING) (default "Number"))
  (slot str_value (type STRING) (default "")) (9)
  (slot value (type FLOAT) (default 0.000000)))
(deftemplate ID_MONTHS
  (slot type (type STRING) (default "Number"))
  (slot str_value (type STRING) (default ""))
  (slot value (type FLOAT) (default 0.000000)))
```

Next in the file is the description of the grid data:

```
(deftemplate LOAN_GRID
  (slot type (type STRING) (default "Grid"))
  (slot col_num (type INTEGER) (default 4))
  (slot col_names (type STRING) (default "colnames_LOAN"))
  (slot col_types (type STRING) (default "coltypes_LOAN"))
  (multislot Data0 (default ))
  (multislot Data1 (default )) (10)
  (multislot Data2 (default ))
  (multislot Data3 (default ))
)
```

It is not among the class attributes as it contains only the results of a calculation, but still has to be defined with its own set of facts. The grid has four columns, which are all numbers, and is defined with a fact similar to (3) where the default value of the “col_num” slot is “4” and the default values of col_types and col_names slots are set to the names of the following two named fact declarations:

```
(colnames_LOAN "No" "Amortization payment" "Yield payment" "Total payment")
(coltypes_LOAN "NUM" "NUM" "NUM" "NUM")
```

2.2. Facts describing the user interface generated for the ontology class

The generated user interface as shown on Fig. 2, contains four text boxes, a combo box, three edit fields and a grid for showing the results of the calculation. The text boxes are defined with a static text definition declaration similar to (5) and the other controls are defined with definitions similar to (4), but the “variableID” and “type” slots are set to the appropriate template fact names and UI control types.

2.3. Rules for attribute processing

In the example simplified CLIPS program, a single rule is introduced that uses the four input fields on the generated user interface to fill the result table. The rule starts with a left-hand side that initializes several CLIPS variables with the values stored as “value” slots of the four class attributes - LOAN_TYPE, ID_BALANCE, ID_YIELD and ID_MONTHS. Also, the LHS initializes a variable that will contain the address of the table that will receive the results of the calculations - LOAN_GRID:

```
(defrule CalculateLoan
(LOAN_TYPE (value ?loanType))
(ID_BALANCE (value ?balance))
(ID_YIELD (value ?yield))
(ID_MONTHS (value ?months))
?grid<-(LOAN_GRID)    =>
```

The right-hand side consists of a CLIPS code that initializes multifield variables that will be the resulting columns of the table. They are filled with data by calculating proper values for the loan monthly payments and in the end the table fact is modified by setting its four columns. Only the code used for generating the example on Fig. 2 is shown – for calculating a regularly amortized loan – one that has equal amortization payments every month. The other cases of the switch statement are withheld.

```
(bind $?month (create$)) (bind $?amoPay (create$))
(bind $?yieldPay (create$)) (bind $?totalPay (create$))
(bind ?repayment (/ ?balance ?months))

(loop-for-count (?crrPer 1 ?months) do
(bind $?month (insert$ $?month ?crrPer ?crrPer))
(switch ?loanType
(case 0 then ... )
(case 1 then
(bind ?remAmount (- ?balance (* ?crrPer ?repayment )))
(bind $?amoPay (insert$ $?amoPay ?crrPer ?repayment ))
(bind $?yieldPay (insert$ $?yieldPay ?crrPer (* ?remAmount (/ ?yield 1200))))
(bind $?totalPay (insert$ $?totalPay ?crrPer (+ ?repayment (* ?remAmount (/ ?yield 1200))))))
(case 2 then ... )))

(modify ?grid (Data0 $?month )(Data1 $?amoPay)(Data2 $?yieldPay )(Data3
$?totalPay )))
```

CONCLUSIONS AND FUTURE WORK

The described way for representing ontology classes by a CLIPS program is used in a framework for developing information systems that has desktop and internet versions. Currently, it is used for creating financial analysis systems, like bank customer evaluation, asset and liability management, market, credit and operational risk assessment systems, regulatory compliance systems and others. All of them are based on CLIPS scripts like the one described in the current publication. The approach allows quick developing of such systems and currently there are over 100 CLIPS programs defining ontology classes in the field of financial analysis. The developed information system builder can be easily and quickly suited to any other field if a detailed ontology is available.

Further development of the CLIPS program could involve avoiding constant control

coordinates and sizes as it is an outdated way for defining user interface. A visual editing tool exists, that makes it easier for CLIPS program developers to create the UI definition and fact template declarations. This allows developers to concentrate on specifying the rules in the program – its business logic. The system can be extended further by adding a tool to import ontologies already defined in standardized ontology languages like OWL. Such a tool can generate the required fact definitions thus vastly broadening the field of use of the information system builder.

REFERENCES

- [1] H. El-Ghalayini, Mohammed Odeh, Richard McClatchey and Tony Solomonides. «Reverse engineering ontology to conceptual data models, » in Proc. Databases and Applications. 2005, pp. 222-227.
- [2] F. Fonseca and J. Martin. «Learning the differences between ontologies and conceptual schemas through ontology-driven information systems, » Journal of the Association for Information Systems, vol. 8, 2007
- [3] T. Gruber. «Ontology, » in the Encyclopedia of Database Systems, Ling Liu and M.Tamer Özsu,Ed.New York: Springer-Verlag, 2009, pp. 1963-1965
- [4] N. Guarino. «Formal ontology and information systems, » in Proc. FOIS'98, 1998, pp. 3-15
- [5] P. Hoefferer. «Achieving business process model interoperability using metamodels and ontologies, » In Proc. ECIS, 2007
- [6] M. Jarrar, J. Demey, R. Meersman. «On using conceptual data modeling for ontology engineering, » Journal on Data Semantics, pp.185-207, Oct. 2003
- [7] S. Nikolov and A. Antonov. «Framework for building ontology-based dynamic applications, » in Proc. CompSysTech, 2010, pp. 83-88
- [8] S. Nikolov. “Storing data of ontology-based dynamic applications,” in Proc. Sixth International Scientific Conference Computer Science, 2011, pp. 134-139
- [9] M. Scott. Programming language pragmatics, Morgan Kaufmann, 2006
- [10] P. Spyns, R Meersman and M. Jarrar. «Data modelling versus ontology engineering, » ACM SIGMOD vol. 31, Dec. 2002, pp. 12-17
- [11] S. Trifonova and S. Nikolov. „XML presentation of financial instruments “, in Proc. Unitech, 2010
- [12] Y. Wand and R. Weber. «Research commentary: information systems and conceptual modeling-a research agenda» Info.Sys.Research, Dec.2002, pp.363-376
- [13] Y. Wand and R. Weber. “An ontological model of an information system,” IEEE transactions on Software Engineering, vol. 16, 1990
- [14] B. Yildiz and S. Miksch. «Ontology-driven information systems: challenges and requirements» in Intl. Conf. on Semantic Web and Digital Libraries, 2007

ABOUT THE AUTHOR:

Samuil Nikolov
Eurorisk Systems Ltd.
31, General Kiselov Str.
9002 Varna, Bulgaria