Mathematical Problems of Computer Science 33, 187–195, 2010.

Developing Java Software for Representation, Acquisition and Management of Strategy Knowledge

Zaven Naghashyan[1] and Edward Pogossian[1,2]

[1]State Engineering University of Armenia
[1,2]Institute for Informatics and Automation Problems of NAS of RA
e-mail: Insl@sci.am, david@dm-lab.sci.am
url: http://dm-lab.sci.am

Abstract

Technical solutions for representing and managing concepts of strategy knowledge for Java based software package and the framework of personalized and communalized knowledge acquisition is described. The following six units are considered - Game Tree Organization, Instances Matching Oriented Classes, Dynamic Class Hierarchy Management, Web Based Graphical Interface and Relations Builder Procedure as well as ideology of knowledge learning and evaluation.

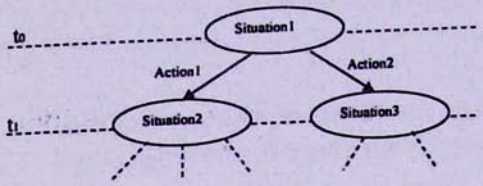
The solutions develop and enhance classical object oriented approaches to be used for representation and management of the concepts of strategy knowledge, their learning and evaluation.

1. Introduction

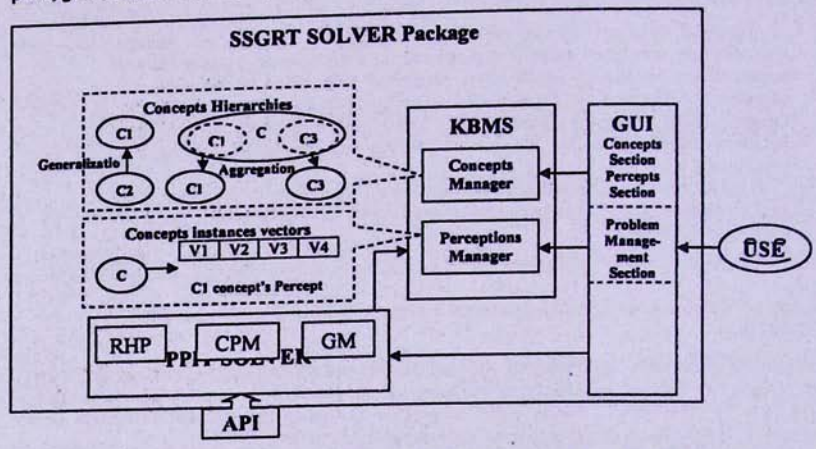
We aim to progress towards developing of software package for solving optimal strategy provision problems where the space of hypotheses of solutions can be specified by reproducible game trees (SSRGT) [1,9-12]. The SSRGT Solver software package continues the practical realization and developments in [1,9-12]and is designed to meet to two main requirements:

1. Users should have a possibility to model an arbitrary problem from SSRGT class within computer environment using the functionality it provides.
2. The software package should contain functional blocks that will be able to solve the already modeled problems.

The problems of SSRGT class meet, particularly, the following requirements [9]: there are: (a) interacting actors (players, competitors, etc.) performing (b) identified types of actions in the (c) specified moments of time and in the (d) specified types of situations and there are identified benefits (goals) for each of the actors. The aggregation of situations represents the game tree, nodes of which are situations themselves and edges are the corresponding transformation actions of the interacting actors.



SSRGT problems are solved for the actor A and for the given situation x if a GT(x, A) strategy is induced which represents all the possible performance trees of the strategies from initial situation x [9]. Thus, for solving an arbitrary problem from SSRGT class the SSRGT solver package should be able to induce the corresponding strategies for the given problem. In [1,10] the PPIT (Personalized Planning and Integrated Testing) Solver module of the package that implements PPIT algorithm has been developed. For inducing the strategies PPIT uses an approach that relies on the experts' problem domain knowledge represented as plans, goals, concepts [9,12] acquired and stores them in the special storage unit.



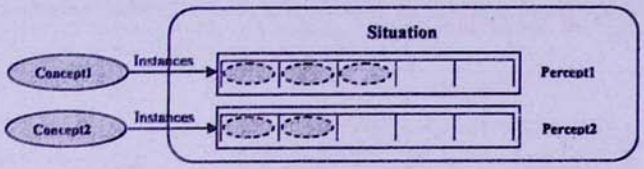
In the current stage the package is implemented with usage of Java programming language and has a modular architecture. The main modules are:

- **PPIT Solver** is a central module of the system and is responsible for solving the already modeled SSRGT problems. It uses functionality of other modules for getting information about the state of the game, and for getting expert knowledge acquired by the system.
- **Concepts Manager** is responsible for storing, organizing and managing expert knowledge represented in the form of concepts – internally represented as Java classes used for instantiating and matching attributes of problems' situations.
- **Percepts Manager** is responsible for storing, organizing and managing instances of concepts, percepts.
- **Graphical User Interface** provides a web based graphical interface that allows users of the package visually configure the package, insert an arbitrary problem from SSRGT class and organize knowledge insertion process in the regular way by using visual instruments including the concepts representation graphical language.

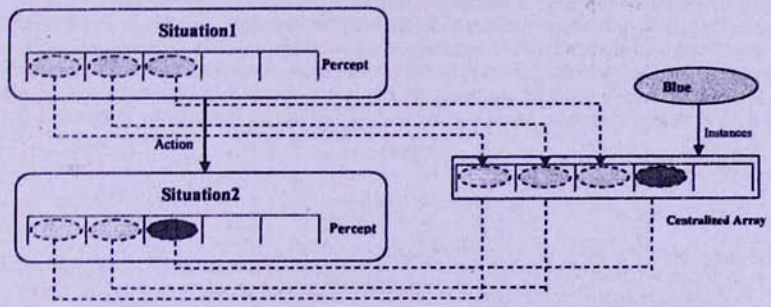
In the rest of the article we will describe the problems that we have encountered during the implementation of the software package requiring notable technical solutions. Each chapter bellow describes a single problem and the corresponding technical solution.

2. Game Tree Organization

The game tree is one of the most important components of representation of SSRGT class problems. Thus, during game tree modeling process we have encountered the problem of the game tree representation within computer memory. Each node of the tree is a certain situation of the problem. Depending on the problems' domains humans perceive situations differently. For example, in case of chess the situations appear as a chessboard with all the fields and figures on it. In case of the management within oligopolies [9] problems situations consist from different parameters that describe market state – levels of supply and demand, level inflation, etc. By the functional modules of the SSRGT package situations of different problems are being perceived and represented in the unified way – as collections of instances of certain classes that were inserted by the users of the package or were formed during the program's life time. We call those classes '**nucleus concepts**', i.e. concepts that are given a priory and are used for initial instantiation of the percepts. Within computer memory each situation can be represented as a data structure that aggregates arrays of instances. For each initial class there is a corresponding array of instances, which are also called '**percepts**'.



Actors affect on situations by executing actions. Actions change states of the components (attributes) of situations, i.e. change values of already existing instances of classes, create new instances and/or delete existing ones. Thus each action transforms situation **s1** to the new situation **s2**. Since we have a requirement to have a complete situations' tree within memory and not just the current situation, the memory should contain several situations. After transformation many components of situations remain the same and only some of instances of initial concepts are changed. Thus we met an important problem of 1) saving memory and not duplicating same instances within different situations 2) fast traversing the tree and not wasting time on dynamically calculating state of percepts for the each node.

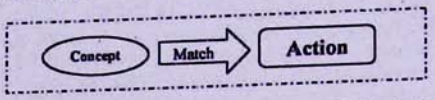


The problem was solved by physically storing instances of initial concepts within centralized arrays and not directly within different situations. In this case percepts within situations will contain just references to the objects within centralized arrays. So, different situations can refer to the same instance of

unchanged components. And for changed attributes new instances will be created and references will be stored within the corresponding percept of the corresponding situation. This approach has many similarities with the well known **copy-on-write** algorithm.

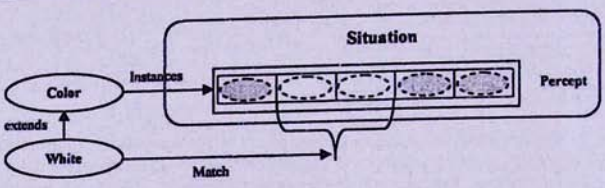
3. Instances Matching Oriented Classes

Strategies are basic units of knowledge in problems of SSRGT class [9]. Other knowledge types like plans, goals, etc. can be used by the PPIT solver module for improving the performance of the strategies [9,12]. All these types of knowledge contain situations or descriptions of situations' components. For example, strategies can be described by rules and the left parts (conditional part) of the rules are descriptions of situations.

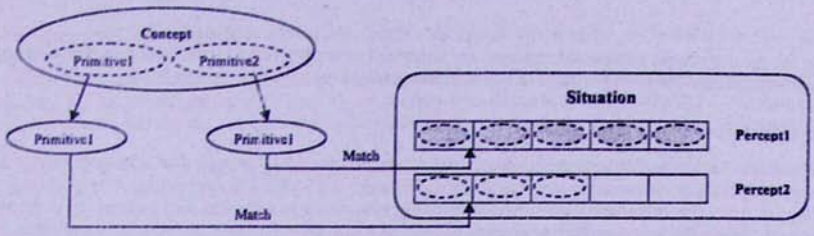


Below we will give more specific definition for the term **concept**.

Classes within knowledge base that are used for describing situations and/or their components are called **concepts**. Concepts either extend the initial concepts or aggregate them as attributes. In contrast to the initial concepts they are not intended to be used for instantiation of objects. Their role is matching instances of the initial concepts.

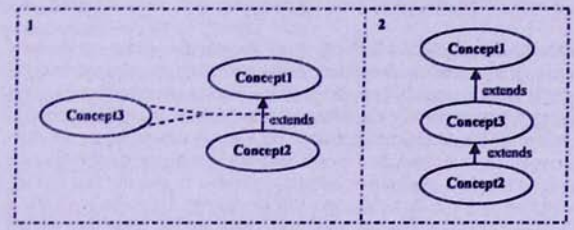


Classical object oriented languages also provide mechanisms to use classes for matching instances. For example there is an "instance of" operator in Java [5]. But matching is being done by usage of direct links between objects and classes that were created during objects instantiation. This means that if there are existing instances for all the attributes of composite class but the class itself wasn't instantiated then it will not be matched over existing objects. But that is wrong, because if the instances of all attributes exist than their composition should represent an instance of the composite class. For overcoming this problem a new instance matching procedure has been developed. The concept matching is implemented recursively by attempting to match all the attributes of the original concept. Attribute can either be again a composite concept or can be a primitive one – an initial concept, or a concept that derived from the initial concept. And because primitive concepts can be matched by direct links or by value ranges, the concept matching will be a recursive attribute searching process and will not be restricted by matching direct links.



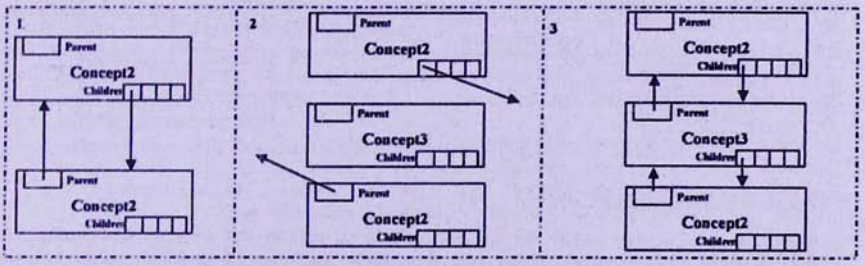
4. Dynamic Classes Hierarchies Management

The SSRGT Solver package is designed to organize the expert knowledge acquisition process in the iterative, regular way. The classe hierarchies that represent concepts relations should be dynamically updated and extended during the program's life time. Classical OO languages support such approach in a very poor way. For example, in Java new classes can be loaded during the program running process but new loaded classes can be added only to the leaves of the existing class hierarchies. This means, that if the class Surgeon extends the class Human and if we want to load dynamically the class Doctor it can't be inserted between Human and Surgeon, in other words Surgeon can't extend it. But during learning process such kind of hierarchy changes are mandatory.



There are several researches for this area where approaches are developed for adding dynamism to the Java class hierarchies [3, 4].

Concepts in the knowledge base are represented as objects of Java classes; this means that Java is used as a meta-language for representing our classes. Java classes and objects are used for implementing classes of objects with semantics that is different from the semantics of original Java language components. This approach allows using dynamic links between classes and adding new classes or removing existing ones.



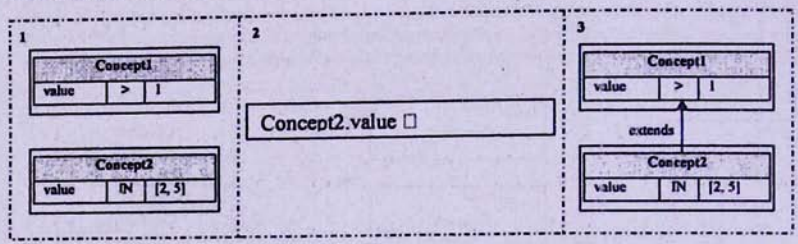
Within the generalization hierarchies concepts are organized as doubly-linked lists. New concepts adding and existing concepts removing is done in a similar way as items of linked list are added or removed [8].

5. Web Based Graphical Interface

Concept acquisition includes dynamic class generation as a sub-process. For each new acquired concept a corresponding class is dynamically generated. As a first stage the SSRGT Solver package supports only an approach when the users are responsible for inserting new concepts (classes). For making the process user friendly graphical user interface with visual tools that allow organizing the knowledge insertion as a regular process has been developed. Most of modern visual programming environments utilize the UML (Unified Modeling Language) for representation of classes [7, 8]. Initially we considered the possibility of the usage of existing visual programming tools as a module of the package that will be responsible for organizing dialog between the user and the program. But, universality of UML brought difficulties for representing special classes – concepts. Because of that, we developed a graphical language that is both similar to UML and allows emphasizing peculiarities of concepts. The graphical language is described in the [2]. For providing flexibility to the users we developed the graphical interface as a web based system. This approach allows using its functionality only by using web browser – without any additional software installation. Also concepts are saved by the web server in the centralized way, which allows the usage of the same system by several users independently from their location.

6. Relations Builder Procedure

By insertion of new concepts (teaching) via graphical interface, users specify the name of the new concept, its attributes and also the concept's relation with existing ones. For example, by inserting the concept Doctor the user can mention that Doctor extends from the concept Human. But sometimes users can forget to mention the corresponding relations or they can add a wrong relation. As consequence of this the knowledge base potentially can contain incorrect information. For solving this problem, we have developed a special program that is executed as a separate process within the system. It traverses the concepts hierarchies and adds the required relations or removes inconsistent ones. It uses the fact that the set of instances of the child concept should be a sub-set of instances of the parent. Depending on this it checks whether the all attributes of a potential parent concept have corresponding attributes that are the same or extends them in the potential child concept. This process continues recursively until the primitive concepts. For the primitive concept the generalization relation can be checked depending on the value ranges of their numeric attributes.



7. Forming Personalized Content

Concepts and attributes that specify the games have original utilities and are used for creating new learnable expertise with the corresponding induced utilities. The required operations can be common

(like geometric transformations preserving some utilities) or be specific for individuals and their experience.

New learnable knowledge can specify winning positions and strategies only with some degree of likelihood. That is because the utilities of expert strategy knowledge are mainly tested by static and quasi dynamic means and are not substantiated by game tree complete dynamic analysis due of their enormous size.

The utilities of expert knowledge stay, as a rule, uncertain what is an essential argument against their totally unified models. In fact, they are mostly individualized with a part of common constituents.

The content of a unit of vocabulary (UV) [12] is a procedure recognizing definite set of realities unified by the name of that UV. That procedure is the solution of a classification problem which is either an acquisition of existing ones or is learned by solving classification problems, specified, as a rule, inductively by providing examples of situations with associated utilities. Thus, the starting point to learn the content is the utility u(P*) we gain in a situation P*.

We associated attributes, concepts other procedures with the P* that for new situations cause the same utility u(P*) if they take place for them.

The procedures associated with utility u(P*) determine corresponding goals which in their turn induce new goals that are causally connected with the old ones, etc. As the result the causality chains are formed where the subordinated child ones become favorite goals to approach to the parent ones.

How to determine the utility u(P) of a P situation given the u(P*) for situation P* with attributes X*= x1,..., xn true for P* ?

Typical rules to form the causality transitive chains include, particularly, the following ones: any P has utility u(P) as close to the u(P*) as many attributes from X* are true for P. As many situations of some class have the same true attributes so much is the likelihood that a new situation with the same true attributes will belong to the same class

There are several methods to form the content of base elements of a vocabulary given training sets in attribute representations. They include methods to form partial Boolean functions; machine learning and statistics based methods, the Beckon-Mile induction, the G. Polya's plausible reasoning, Piaget's assimilation-accommodation, and etc. methods. If training examples are unique it is appropriate to use transformations preserving the utility of those examples which can be, particularly, problem dependent, common geometrical or logical, etc.

8. Proving adequacy of models

Proving adequacy of UER chess solutions, in general, can be identified as a computationally hard problem of identification or proving equality of programs. Thus, there is no straightforward and complete testing for UERchess solutions.

In general, the dialogue has to be organized between carriers of the concepts where the experts must test the adequacy of models. The testing has to be on a regular, independent from a particular subject base. There have to be criteria and technique to estimate the level of approximation, make necessary measurements, interpret and substantiate the experiments requiring, particularly, not an abstract, absolute approximation of some ideal concepts but only coincidence with the expert ones

We test the models of expertise as the following:

- examples are generated on the board by the models and commented by the experts; for that a generator using models of the concepts as inputs generates situations on the board that don't contradict to the models, submits them to the experts and gets their estimates for further possible corrections;
- because there is a threat that only a part of the meaning of the concepts was "caught " by the models they have to be tested by complementary means like the following:
  - experts generate examples which are tested by corresponding models
  - situations are taken from some repository and the reactions of the models and experts are compared
  - situations are taken from the professional books or hand-books

* there is an evidence of completeness of the set of tests [on-the- job performance ideology]

Summarizing, the following consequences of the UCR study can be formulated.

1. Concept type UCR are, as a rule, hypothetic, approximate descriptions of classes of winning by Zermelo positions [9] , or are constituents of such descriptions.
2. The plan and strategy types of UCR can be interpreted as constructions and elements based on the constituents of the game tree.
3. The likelihood of hypothetic descriptions, as a rule, is small because precise determination whether the positions are winning cannot be done since it needs generation and evaluation corresponding strategies what is a hard computational problem. Therefore, the value of concepts, in principle, cannot be common for all experts and descriptions of concepts along with common constituents have to include an essential part of subjective ones.
4. The ideology of the Object Oriented (OO) programming along with OO Data Bases, Case Based Systems (CBS), Neuron Nets, ID3 and Boolean functions reconstruction techniques (Dhar, Stein,97;Russel, Norvig, 2002) seems appropriate to integrate common and subjective constituents of concepts.
5. The techniques equally with CBS can be corresponded with human mechanisms of unconscious and intuitive knowledge processing with a huge amount of precedents.

Conclusion

The article presents technical solutions that were implemented in the SSRGT Solver software package as well as describes the framework of strategy knowledge learning and evaluation. The solutions concern different aspects of the management of concepts and their instances. We define concepts as special OOP classes that are used by the SSRGT Solver for instantiation of situations' components and/or for matching situations by matching their components. Bellow is the list of solutions: Game tree organization, Instance matching oriented classes, Dynamic class hierarchy changes, Web based graphical interface, Relations builder procedure.

The expert knowledge represented by concepts and used by the software package can be divided into two categories. *Communicable knowledge* - represents understandable by the members of some community knowledge, while the *personalized one* represents created by individuals of the community using their personal experience and perception of the matter being represented. In the current stage all the technical developments concern to the parts of the package that support a communicable portion of the knowledge.

As a next phase of our development we consider implementation of technical approaches to acquire, store, organize and manage personalized portions of expert knowledge.

Acknowledgement

We would like to express our gratitude to Artem Harutyunyan, Emil Matevosyan and Andranik Khandanyan for their constructive comments and support.

References

1. E. Pogossian, V. Vahradyan and A. Grigoryan, "On Competing Agents Consistent with Expert Knowledge", *Lecture Notes in Computer Science, AIS-ADM-07: The International Workshop on Autonomous Intelligent Systems - Agents and Data Mining*, St. Petersburg, Russia, pp. 229-241, June 6-7, 2007.
2. T. Bagdasaryan, A. Grigoryan and Z. Nagashyan, "Developing of a scripting language interpreter for regular acquisition of expert knowledge", *International Conference in Computer Sciences and Information Technologies*, pp. 187-191, Yerevan, 2009.
3. Su Lixin, H. Mikko, Lipasti. Dynamic Class Hierarchy Mutation - http://www.ece.wisc.edu/~pharm/papers/cgo4.pdf

4. R. Joel Brandt; Goldman, J. Kenneth, Run-time Modification of the Class Hierarchy in a Live Java Development Environment - http://cse.wustl.edu/Research/Lists/Technical%20Reports/Attachments/642/403_hierarchy.pdf
5. Java Language Specification - http://java.sun.com/docs/books/jls/
6. Jude official site - http://jude.change-vision.com/jude-web/index.html
7. IBM Rational Rose - http://www-01.ibm.com/software/rational/
8. Linked lists basics - http://cslibrary.stanford.edu/103/
9. E. Pogossian, "Combinatorial Game Models For Security Systems", *NATO ARW on "Security and Embedded Systems"*, Porto Rio, Patras, Greece, Aug., pp. 8-18, 2005.
10. Э. Погосян, В.Варадян, А.Григорян, "Эксперименты согласования знаний экспертов с принятием решений в этюдах Рети и Нодареишвили". *Proceedings of IPIA in Mathematics and Computer Sciences*, p. 20, 2007.
11. E. Pogossian, *Adaptation of Combinatorial Algorithms*, (in Russian), pp. 293,Yerevan, 1983.
12. E. Pogossian, Specifying Personalized Expertise. International Association for Development of the Information Society (IADIS): International Conference Cognition and Exploratory Learning in Digital Age (CELDA 2006), Barcelona, Spain, pp.151-159, 2006.

Java ծրագրաշարի մշակում ռազմավարական գիտելիքների ներկայացման, ընկալման և կազմակերպման նպատակով

Զ. Նաղաշյան և Է. Պողոսյան

Ամփոփում

Հոդվածում նկարագրված են Java ծրագրավորման լեզվի հիման վրա ստեղծված ծրագրաշարին կիրառված տեխնիկական լուծումները և անճնավորված ու հասարակայնացված գիտելիքների ընկալման հիմնակմախքը: Դիտարկված են հետևյալ վեց թեմաները՝ խաղային ծառի կազմակերպումը, նմուշների ընտրման համար դասերի մշակումը, դինամիկ դասերի ստորադասային կազմակերպումը, վեր իմտերֆեյսի մշակումը, դասերի միջև հարաբերությունների կառուցման ընթացակարգը ինչպես նաև գիտելիքիների ուսուցման և գնահատման գաղափարները: Կիրառված մոտեցումները զարգացնում և ընդլայնում են դասական օբյեկտորոշված ծրագրավորման մեթոդները, թույլ տալով օգտագործել դրանք ստրատեգիաներում գիտելիքների հասկացությունները համակարգչի միջոցով ներկայացնելու, կազմակերպելու, ուսուցանելու և գնահատելու համար: