Soft Constraints for KnowLang

Ugo Montanari
Dipartimento di Informatica, Università di Pisa
Pisa, Italy
ugo@di.unipi.it

Emil Vassev
Lero–the Irish Software Engineering Research Centre
University of Limerick
Limerick, Ireland
emil.vassev@lero.ie

ABSTRACT

Constraints are widely used in information technologies and research fields such as programming languages, artificial intelligence, databases, information security, web technologies, etc. In this paper, we present our preliminary steps of using soft constraints for knowledge representation. We integrate soft constraints in KnowLang, a formal language for knowledge representation in self-adaptive systems. KnowLang allows for efficient and comprehensive knowledge structuring where ontologies are integrated with rules and Bayesian networks. The approach targets at a technique where knowledge can be represented as special restrictive rules that may require full or partial satisfaction, i.e., restrictions are represented as some sort of good-to-have properties.

Categories and Subject Descriptors

D.3.2 [**Programming Languages**]: Language Classifications—*Very high-level languages; Multiparadigm languages;*; I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program synthesis; Program transformation;*; D.2.11 [**Software Architectures**]: Domain-specific architectures; Languages

General Terms

Languages, Theory, Design

Keywords

knowledge representation, soft-constraints, KnowLang

1. INTRODUCTION

Computer intelligence mainly excels at formal logic, which allows it, for example, to find the right chess move from hundreds of previous games. Intelligent systems might employ appropriately *structured knowledge* that is used by embedded *inferential engines*. Software engineers use knowledge representation (KR) techniques [11] to structure large amounts of knowledge for the purpose of computer intelligence. Knowledge representation structures may be primitives such as *rules*, *constraints*, *frames*, *semantic networks*, *concept maps*, *ontologies*, and *logic expressions*. Whatever knowledge primitives they use, engineers must structure computer knowledge so that the system can effectively process it and humans can easily perceive the results.

In general, the so-called *soft constraints* [3, 2, 7] might be used as a KR technique that will help designers impose *constraining requirements* for special *liveness properties* of an intelligent system. In this context, the term *liveness property* must be considered as an approximation to our understanding of a good-to-have property. In this paper, we elaborate on the theory of soft constraints to build a model for KR of desired restrictions on values held by the variables of the system in question.

The rest of this paper is organized as follows. Section 2 briefly presents KnowLang, defines some terminology and presents the so-called constraint satisfaction problem. Section 3 presents our approach to using soft constraints in KnowLang. Section 4 represents a case study problem based on a mobile robotics platform and, finally, Section 5 provides brief concluding remarks and a summary of our future goals.

C^3*S*^{2}*E-12* 2012, June 27-29, Montreal, [QC, CANADA]
Editors: B. C. Desai, S. Mudur, E. Vassev

2. BACKGROUND

2.1 KnowLang

KnowLang is an initiative undertaken by Lero - the Irish Software Engineering Research Center - within Lero's mandate in the ASCENS project. Autonomic Service-Component ENSembles (ASCENS) [1] is an FP7 (Seventh Framework Program [8]) project targeting the development of a coherent and integrated set of methods and tools providing a comprehensive approach to developing ensembles (or swarms) of intelligent, self-aware and adaptive service components. One of the main scientific contributions that we expect to achieve with ASCENS is related to *knowledge representation and reasoning*. A key feature of KnowLang is a multi-tier specification model (see Figure 1) allowing for integration of ontologies together with rules and Bayesian networks [10]. The language aims at efficient and comprehensive knowledge structuring and awareness based on logical and statistical reasoning. It helps us tackle 1) explicit representation of domain concepts and relationships; 2) explicit representation of particular and general factual knowledge, in terms of predicates, names, connectives, quantifiers and identity; and 3) uncertain knowledge in which additive probabilities are used to represent degrees of belief. Other remarkable features are related to knowledge cleaning (allowing for efficient reasoning) and knowledge representation for autonomic robotic behavior. KnowLang [12, 13] imposes a *multi-tier specification*

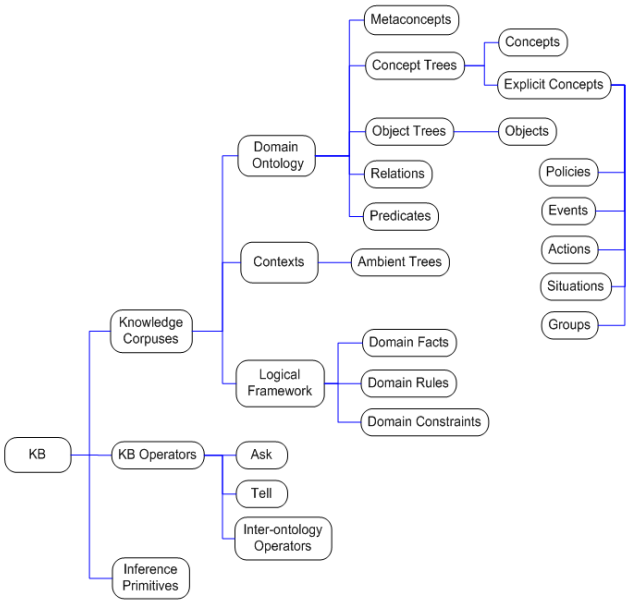

Figure 1: KnowLang Multi-tier Specification Model

model (see Figure 1)[13], where we specify knowledge corpuses, KB (knowledge base) operators and inference primitives at different hierarchically organized tiers. As shown in Figure 1, knowledge is organized in a special Knowledge Base (KB) at three main tiers:

1. Knowledge Corpuses;
2. KB Operators;
3. Inference Primitives.

The tier of Knowledge Corpuses is used to specify KR structures. The tier of KB Operators provide access to Knowledge Corpuses via a special class of *ASK* and *TELL operators*, where ASK operators are dedicated to knowledge querying and retrieval and TELL operators allow for knowledge update. Moreover, this tier provides for special *inter-ontology operators* intended to work on one or more ontologies. Note that all the KB Operators may imply the use of Inference Primitives, i.e., new knowledge might be inferred and eventually stored in the KB. The tier of Inference Primitives is intended to specify algorithms for inference and reasoning. In this paper, we do not present the language itself, but the interested reader is advised to refer to [12, 13] for more information on the KnowLang's specification model.

2.2 Definitions

With the notion of Soft Constraint for KnowLang (SCKL), we intend to associate tuples of possible values held by special KnowLang "variables" with possible preferences. Thus, to express a SCKL, we consider:

1. a tuple of variables, representing a part of the system expressed with KnowLang;
2. a preferred combination of values held by these variables;
3. special constraint conditions defining conditions when the "preferred combinations of values" must be held.

In this approach, the notion of "variable" is closely related to the notion of KR symbol. SCKL variables (called KR Variables) represent concepts defined by the ontologies (e.g., "Policy" or "Action"), concept properties, or relations. Consecutively, the notion of "value" is associated with realization of a concept (e.g., an object), realization of a concept property, or realization of a relation. SCKL Values are also called KR Values. Note that SCKL will not be used to set what values are allowed, but rather at what specific state (component state or global system state) or situation are allowed.

2.3 Constraint Satisfaction Problem and Soft Constraints

The classical *constraint satisfaction problem* (CSP) framework [9] is a well-known paradigm, that is suited to specify many kinds of real-life problems and that has been broadly investigated in computer science and artificial intelligence. The key idea underlying CSP is to solve a problem by stating constraints representing requirements about the problem and, then, finding solutions satisfying all the constraints.

A CSP defined over a constraint network consists of a finite set of variables, each associated with a domain of values and a set of constraints. A constraint, or even a network of constraints, states the legal combinations of values of a certain subset of variables. Formally, constraints are functions which, given an assignment of the variables to some domain, return a Boolean value. In this sense, constraints are declarative, namely they specify which relationship must hold, while disregarding the computational procedure to enforce that relationship. A solution to a CSP is an assignment to each variable of a value from its domain such that all the constraints are satisfied.

Classical CSPs are not well-suited in several real-life scenarios. Indeed, CSPs are not able to model constraints that are preferences rather than strict requirements or to provide a "non-complete" solution when the problem is over-constrained. Basically, this is the idea behind the notion of "soft constraints". Roughly, a soft constraint is a constraint that rather than returning a Boolean yields more informative values such as a preference value or a cost. When combining constraints, one has to take into account such additional values.

Some extensions of *classical* CSPs give specific interpretations of soft constraints like *weighted CSPs* for modelling cost functions, *probabilistic CSPs* or *fuzzy CSPs*. The *semiring-based constraints* [3] are more generic extensions to soft constraints, in the sense that they can model different kinds of constraints by varying their underlying structure.

A *constraint semiring* (c-semiring) [3] is an algebra $\langle A, +, \times, 0, 1\rangle$, where $\langle A, +, 0\rangle$ and $\langle A, \times, 1\rangle$ are commutative monoids, $+$ is idempotent, $\times$ distributes over $+$, 1 and 0 are absorbing elements for $+$ and $\times$ respectively (i.e., $a + 1 = 1$ and $a \times 0 = 0$ for all $a \in A$). C-semirings are also equipped with a partial ordering $\leq$ such that $a \leq b$ *iff* $a + b = b$, which means that a is worse than b, or, more interestingly, that a entails b. Intuitively, the preference level associated to each variable instantiation is modelled as a value of a c-semiring; the combination of constraints is expressed by the product operation, while the sum $a + b$ chooses the worst constraint better than a and b. Moreover, 1 is the maximal and 0 the minimal element. Remarkably, several efficient algorithms defined for ordinary constraints,

like constraint propagation or dynamic programming, can be generalised to c-semirings.

3. SOFT CONSTRAINTS FOR KNOWLANG

Our notion of SCKL internalizes, in the style of [7], the KnowLang variables V into a c-semiring S . Namely, our SCKL constraints are functions associating to each feasible assignement of variables V a value of S. Thus to formally define a SCKL for a constraint semiring S, we consider a set of KR Variables I and a set of possible KR Values V, where for each $i \in I$, there is a set $V_i \subset V$ of possible KR Values for the variable i. A SCKL can be defined as $c := (J; P)$ with $c \in C$ (set of constraints), i.e., c is defined as a pair $(J; P)$ where $J := (j_1, ..., j_k)$ is an ordered subset of I (denoted as $J \subset I$) and P is a function mapping $V_{j1} \times, ..., \times V_{jk}$ into S.

The semiring values are ordered (usually, totally ordered) by the semiring ordering $\leq$. Thus it is possible to identify the preferred variable assignments as those with the highest semiring value associated to them.

Often, SCKLs have additional *constraint conditions* Z, expressed as Boolean operations over the KnowLang Ontologies O. Normally, a condition shall be expressed with "states", "situations", "events", and/or "actions".

Once KnowLang variables and their possible values are fixed, different SCKLs are characterized by their functions P. Thus the semiring operations of addition and multiplications can be extended to SCKLs by composing functions P pointwise.

In addition, an operation of *restriction* can be defined with eliminates a variable assigning to it the best possible value. Technically, the restriction $(x)p(x)$ is obtained by summing up $p(v)$ for all possible values of v. The *projection* operation is restriction's dual: to evidence the effect of a constraint on a set of variables is sufficient to restrict it with respect to all the other variables.

Complex *SCKL systems* can be obtained by applying the above operations to elementary (typically finite, explicitly listed) SCKLs. Often, several elementary SCKLs involving only a few variables are multiplied, obtaining a large *constraint network*. Then the resulting complex SCKL is projected into some variables, which represent the visible interface of the system.

A *Constraint Satisfaction Problem* (CSP) for a SCKL system is to find an assignment of its variables which returns the best semiring value. Of course there can be ties, i.e. several assignments can yield the same value, which makes the solution process nondeterministic. The situation is particularly critical for the classical semiring, where assignments can return only 1 or 0, i.e. possible or impossible. If the semiring ordering is partial, a solution is an assignment which returns a semiring value which is non-dominated (i.e. it is a local maximum, sometimes called *Pareto-optimal*).

4. CASE STUDY

KnowLang has been applied to derive an initial KR structures for the marXbot mobile robotics platform [6], one of the ASCENS' case studies. As part of this research, we have investigated the possibility of integrating soft constraints with KnowLang.

4.1 The marXbot Robotic Platform

The marXbot [6] is a modular research robot equipped with a set of devices that help the robot interact with other robots or the robotic environment. The environment is defined as an arena where special cuboid-shaped obstacles are present in arbitrary positions and orientations. Moreover, the environment may contain a number of light sources, usually placed behind the goal area, which act as environmental cues used as shared reference frames among all robots. A marXbot robot is equipped with a set of devices to interact with the environment and with other robots of the swarm, and with a locomotion system to move forward, backward and to turn left and right.

4.2 KR for marXbot Robot

Figure 2 depicts a KR structure called *concept tree*, which is specified with KnowLang. The concept tree has tree root "Thing" represented with the concept "Thing". The latter is determined by the meta-concept "Robot Thing", which carries information about the interpretation of the root concept "Thing" such as "Thing is anything that can be related to the robot". According to this concept tree there are two categories of things in a robot: *entities (physical entities)* and *virtual entities*, where both are used to organize the vocabulary in the internal robot domain. Note that all the explicit concepts (see Figure 1) are presented as concepts in this concept tree - qualified path "Thing->Virtual Entity->Phenomenon", i.e., in this concept tree, the explicit concepts inherit the concepts "Phenomenon", "Function" and "State". The following KnowLang code presents the actual specification of the *Locomotion_System* concept. Due to space limitations, we do not present the language syntax, which can be easily grasped from the example below.

```
CONCEPT Locomotion_System {
  CHILDREN {}
  PARENTS { SC.Thing..System }
  STATES { STATE operational {} STATE on {} STATE off {}  }
  PROPS {
    PROP engine { TYPE {SC.Thing..Engine} CARDINALITY {1} }
    PROP wheel { TYPE {SC.Thing..Wheel}  CARDINALITY {6}  }
    PROP locomotion_soft { TYPE {SC.Thing..Locomotion_Soft }
         CARDINALITY {1} }
    PROP battery {TYPE {SC.Thing..Battery } CARDINALITY {1} }
  }
  FUNCS {
    FUNC move {
      TYPE {SC.Action.Move }
      PRE_CON {} POST_CON {}
      PARAMS { SC.Thing..Direction }
      RETURN {}
      BODY { IMPL }
      ERRORS { } }
    FUNC stop {
      TYPE {SC.Action.Stop }
      PRE_CON {} POST_CON {}
      BODY { IMPL }
      ERRORS { } }
    FUNC turn {
      TYPE {SC.Action.Turn }
      PRE_CON {} POST_CON {}
      PARAMS { SC.Thing..Direction, SC.Thing..Angle}
      BODY { IMPL }
      ERRORS { } }
}
```

Note that, every concept specified with KnowLang has an intrinsic attribute $STATE$ that may be associated with a set of possible state values the concept instances may be in. The $STATE$ attribute is a concept descending from the *State concept* (see Figure 2). A system may occupy a new state when values of concept properties have been changed

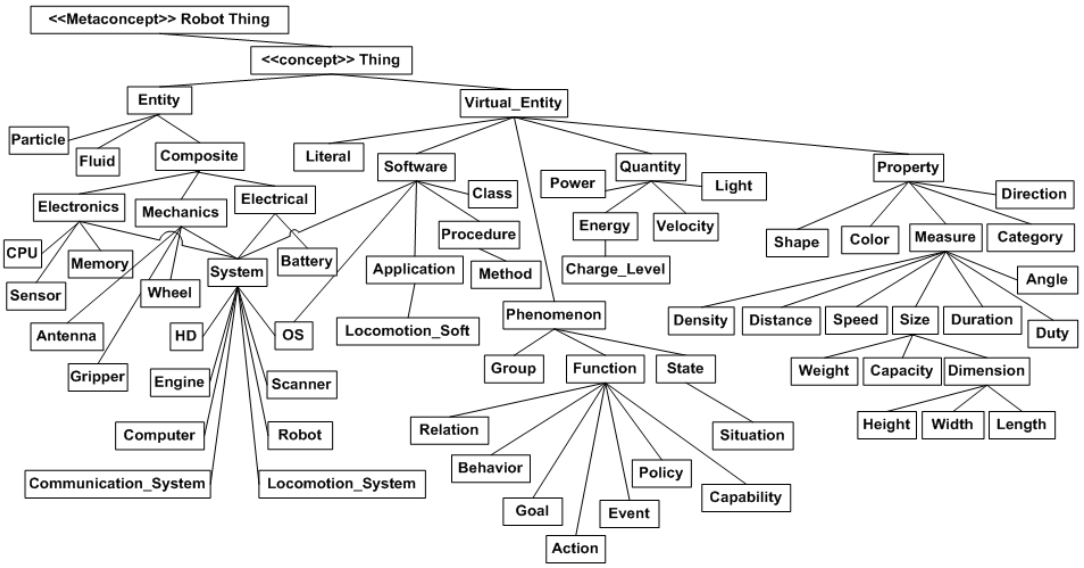

Figure 2: Concept Tree: "Robot Thing"

or some events or actions have occurred in the system or the environment [13]. Therefore, a state can be determined by values held by concept properties, events or actions. Thus, a state of a complex concept might be the product of the states of its properties.

```
Predicate.Is_Operational(THIS.locomotion_system)
```

For example, we may consider the states of the following concept instances: $robot[1]$ and $robot[1].locomotion\_system$. The possible sets of *state values* associated with these states could be:

```
robot[1].STATES := { moving_forward, pursuing_goal_B,
                pursuing_goal_A, operational, on, off }
robot[1].locomotion_system.STATES := { operational, on, off }
```

4.3 Soft Constraints for marXbot Robot

With reference to the KR structures presented in Section 4.2, we consider the following constraint for marXbot.

A marXbot robot has six wheels, each of which can be considered as a variable that can take state values in:

```
wheel.STATES := { clockwise, counterclockwise, stop, idle }
```

The wheels operate in pairs, and on each wheel pair we apply a constraint stating that pair weels must take "certain" values. In fact, if two wheels in a pair are turning clockwise and anticlockwise respectively the pair is moving forward. If in a pair a wheel is turning clockwise and the other wheel is stopped, then the pair turns left, etc. Here, the tree pairs of wheels can have global state values as following:

```
wheel_pair.STATES := {forward, backward, left, right, stop, idle}
```

The soft constraint approach considers a pair of wheels equipped with three variables representing left and right wheels and the global state of the pair:

```
pair(global) :- left(global,leftwheel),right(global,rightwheel)
```

where the constraints $left$ and $right$ determine the relation between the possible states of the wheels and the global state. For classical Horn clauses, the relational operator ":-" means that the right hand side implies the left hand side, while for all the soft constraints ":-" means that the right hand side is larger than the left hand side in the semiring partial ordering. The comma "," means multiplication, logical AND in the classical case. Here, for a pair, $left$ and $right$ are defined as:

```
left(forward,counterclockwise) :-
right(forward,clockwise) :-
left(backward,clockwise) :-
right(backward,counterclockwise) :-
left(left,stop) :-
right(left,clockwise) :-
left(right,counterclockwise) :-
right(right,stop) :-
left(idle,idle) :-
right(idle,idle) :-
left(stop, stop) :-
right(stop,stop) :-
```

Here the empty right hand side obviously means 1. For instance, the assignment $global := forward$, $leftwheel := counterclockwise$ and $rightwheel := clockwise$ is allowed, namely it is mapped to 1. Conversely, no assignment is possible with $leftwheel := clockwise$ and $rightwheel := clockwise$.

The robot has three pairs of wheels and a GLOBAL state variable with possible values:

```
robot[1].STATES := {forward, left, right, stop}
```

Thus:

```
robot[1](GLOBAL) :- OK(GLOBAL,global1,global2,global3),
                    pair(global1),pair(global2),pair(global3)
```

where OK is defined as:

```
OK(forward,forward,forward,forward) :- very fast
...
OK(forward,forward,idle,idle) :- slow
OK(left,left,idle,idle) :- slow
...
```

Notice that here OK has been defined in a soft way: the option where the forward action of the robot is obtained with all three pairs contributing to it has been evaluated as "very fast", assuming that the semiring has such a value, while if only one pair has forward and the other two have idle, the value is "slow". Here the idea is that "very fast" is better in the semiring ordering than "slow".

5. CONCLUSION

This paper has presented an approach to integrating soft constraints with KnowLang, a formal language for knowledge representation in autonomic and self-adaptive systems. KnowLang implies a multi-tier specification model that allows for integration of ontologies together with rules and Bayesian networks. The described approach enriches the language with a technique where knowledge can be represented as special restrictive rules that may require full or partial satisfaction. Restrictions are represented as some sort of good-to-have properties, e.g., special liveness properties. In our approach, soft constraints can be employed to enhance Constraint Satisfaction Problems, where soft constraints give weights to variable assignments. However, soft constraints can also be used to extend constraint logic programming [4], and concurrent constraint programming [5]. In both cases, constraints enhance the question answering capability of the logic, clause-based approach. Uniformity of the underlying constraint storage for modeling data components is a valuable asset. A case study problem has been presented to demonstrate the utility of the approach. Note that KnowLang is still under development as part of the ASCENS international European project [1]. Our plans for future work are mainly concerned with further and complete development of KnowLang including full integration of the soft-constraint technique with the language.

6. ACKNOWLEDGEMENTS

This work was supported by the European Union FP7 Integrated Project Autonomic Service-Component Ensembles (ASCENS) for both authors and by Science Foundation Ireland grant 03/CE2/I303_1 to Lero-the Irish Software Engineering Research Centre at University of Limerick, Ireland for the second author.

7. REFERENCES

[1] ASCENS. ASCENS - Autonomic Service-Component Ensembles, 2010. `http://www.ascens-ist.eu/`, last viewed April 2012.

[2] A. Biso, F. Rossi, and A. Sperduti. Experimental results on learning soft constraints. In *Proceedings of KR'2000*, pages 435–444. SRI International, Menlo Park, California, USA, 2000.

[3] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[4] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint logic programming: Syntax and semantics. *ACM Trans. Program. Lang. Syst.*, 23(1):1–29, 2001.

[5] S. Bistarelli, U. Montanari, and F. Rossi. Soft concurrent constraint programming. *ACM Trans. Comput. Log.*, 7(3):563–589, 2006.

[6] M. Bonani, V. Longchamp, S. Magnenat, P. Retornaz, D. Burnier, G. Roulet, F. Vaussard, H. Bleuler, and F. Mondada. The marxbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*. IEEE/RSJ, 2010.

[7] M. G. Buscemi and U. Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In R. D. Nicola, editor, *ESOP 2007, LNCS 4421*, pages 18–32. Springer, 2007.

[8] European Commission - CORDIS. Seventh Framework Program (FP7), 2012. `http://cordis.europa.eu/fp7/home_en.html`, last viewed April 2012.

[9] U. Montanari. Networks of constraints: Fundamental properties and application to picture processing. *Information Science*, (7):95–132, 1974.

[10] R. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.

[11] E. Vassev and M. Hinchey. Knowledge representation and reasoning for intelligent software systems. *IEEE Computer*, 44(8):96–99, 2011.

[12] E. Vassev and M. Hinchey. Towards a formal language for knowledge representation in autonomic service-component ensembles. In *Proceedings of the 3rd International Conference on Data Mining and Intelligent Information Technology Applications (ICMiA 2011)*, pages 228–235. NASNIT,ICMIA, IEEE, 2011.

[13] E. Vassev and M. Hinchey. Knowledge representation for cognitive robotic systems. In *Proceedings of the 15th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing Workshops (ISCORCW 2012)*, pages 156–163. IEEE Computer Society, 2012.