Journal of Intelligent & Fuzzy Systems 27 (2014) 655–666
DOI:10.3233/IFS-131097
IOS Press

Smart experience engineering to support collaborative design problems based on constraints modelling[1]

Alejandro Cálad-Álvarez[a,*], Ricardo Mejía-Gutiérrez[a], Cesar Maldonado Sanín[b] and Edward Szczerbicki[c]
[a]*Design Engineering Research Group, Universidad EAFIT, Medellín, Colombia*
[b]*School of Engineering, Newcastle University, ES Building, Newcastle, NSW, Australia*
[c]*Gdansk University of Technology, Gdansk, Poland*

Abstract. Engineering design is a knowledge intensive process. Experts' experiences from different product life-cycle stages play a key role in problem solving during design decision making by linking up knowledge to find better solutions for a specific design problem. Different approaches have been used to support Collaborative and Concurrent Product Design, such as Constraint Satisfaction Problem (CSP) modelling. Additionally, due to the need of capitalizing these experiences, the Set of Experience Knowledge Structure (SOEKS) is used to store and manage experiences of decision making process. However it was necessary to adapt the structure to property manage the constraints approach used in Engineering Design. This article presents a proposed adaptation to SOEKS structure in order to be able to store and reuse CSP design experiences for further re-use and analysis of multiple experiences. Finally, a case study is presented in order to show how the proposed approach can be used in the industrial applications.

Keywords: Set of Experience Knowledge Structure, Concurrent engineering, Knowledge engineering, Knowledge representation, Artificial intelligence, Constraint satisfaction

1. Introduction

Specialization of companies in specific disciplines have fostered new ways of work based on responsibilities distribution according to companies' strengths and experience. Due to this, nowadays complex projects have their partners distributed in different places all around the world. Under this environment, distributed teams are built to work together, finding the best integrated solution to a specific problem. This approach has important impact in early design stages of new products. Concepts like Concurrent and Distributed Engineering tackled this issue in a general way, arousing interest in both, research and industrial sectors.

An important concept, closely associated with Concurrent Engineering (CE), is the Product Life Cycle (PLC) which is more and more relevant to knowledge capture and to evaluate the impact of decision making

[1]This document is a collaborative effort between the Design Engineering Research Group (GRID) from Universidad EAFIT (Colombia) and the Knowledge Engineering Research Team (KERT) from The University of Newcastle (Australia).

*Corresponding author. Alejandro Cálad-Álvarez, Design Engineering Research Group, Universidad EAFIT, Engineering Building, Cra. 49 No. 7 Sur 50 050022, Medellín, Colombia. Tel.: +574 2629500/Ext. 9712; Fax: +574 2664284; E-mail: acaladal@eafit.edu.co.

about engineering changes. PLC considers all relevant elements related to the different life cycle stages such as design, manufacture, distribution, use, maintenance and disposal of a product. Taking into account those stages of the PLC become a key issue to design a product. Also, as designers' decisions affect other departments, such as manufacture, storage, transport, commercialization, etc. the design stage must include constraints of all subsequent stages in order to avoid future problems in downstream lifecycle stages.

Therefore, it is necessary to include experts' knowledge from different lifecycle stages (preferably from all of them) and to establish relationships among them, looking to ensure completeness of the design review by linking up their knowledge to find better solutions for a specific design problem. This way of working is known as CE. However, in a collaborative environment, it is difficult to put together knowledge of diverse nature by integrating constraints of different lifecycle stages into a single design proposal that satisfies all of them. Notwithstanding, different approaches have been developed to support Collaborative and Concurrent Product Design [15, 16, 20, 26]. One of them is Constraint Satisfaction Problem (CSP) which has received much interest as it allow designers, not only to model a design problem in terms of dimensional issues, but also in terms of configuration or behavioural analysis. It permits defining discrete and continuous variables associated in varied of constraints that are represented in mathematical expressions. However, CSP also allow designers to include in the same model configuration problems by defining symbolic variables, which are linked by logic constraints.

On the other hand, design is all about decisions, like defining the shape (weight, volume and bulk), the best material for the shape selected, the color, some physical properties, among others. Designers fix the values of these attributes based on constraints from experts, derived from previous, similar, or equal experiences. Using CSP, in early design stages, as a tool for reducing the solutions space (which corresponds to the solution set of product design proposals) is also part of the decision making process in engineering design. The experience necessary to define the components of a CSP model can be called technical or domain-specific knowledge. Therefore, knowledge in a CE environment is an important asset that can be collected, stored and re-used as experiences to improve the design process.

Although Knowledge Representation (KR) has been a topic of studies and different proposals have been made, there is not a unique consensus about one universal representation that can store knowledge for different domains. This is basically due to the fundamentals of the problem from one domain to another, also because the content and the form of the proposed representation in each case [24]. Among all proposals to represent knowledge, there is a knowledge structure that stores and manages experience for decision making processes. It is called Set of Experience Knowledge Structure (SOEKS). As designers make decisions during the definition of a product, SOEKS can be used to represent and to store designers' knowledge in a predefined structure to re-use and help designers while taking decisions regarding the components of a CSP model in the conceptual design stage of a product.

Having represented and stored experts' knowledge, it is possible to make inferences and to help designers during CSP modelling process by suggesting variables and/or constraints based on the component's characteristics or even, the problem context. Once having represented the components of a CSP model under SOEKS' structure, it is also possible to represent a refined and approved CSP model as whole SOEKS and, by this, store the set of the experiences of the complete process.

This paper presents a knowledge modelling approach for engineering design problems based on CSP and using SOEKS as a Knowledge Representation Structure. This way of design knowledge representation allows to share and distribute experiences for enhancing the definition of CSP models applied to engineering design. The structure of the paper is divided in 5 sections. In Section 2, a literature review about CSP and Knowledge Representation Trends are introduced. Section 3 presents the interpretation of CSP syntax, the aggregation of new structures to SOEKS, a change in how to interpret attributes and finally a change in the SOEKS structure. In Section 4 a Case Study is presented and in Section 5, the conclusions and further research are addressed.

2. Literature review

2.1. CSP as knowledge representation in engineering design

For the last years CSP has been commonly used in Engineering Design as a way for representing knowledge. As many design problems can be formulated under this concept, its use grew incrementally to support this particular engineering application [1, 2, 10].

The constraints used during a design process include heuristics, tables, guidelines or computer simulations, as well as machine limitations, packaging limits, among others constraints issued from different experts throughout the PLC. A majority of those limitations can be expressed as mathematical constraints [3] and therefore they may be used in these kind of numerical approaches.

That is why technical knowledge in Engineering Design may be modelled as a set of mathematical relations among a set of variables. Those variables may have a finite set of possible values to take. Therefore, a modelling process exists to represent that knowledge in terms of $P(\mathcal{V}, \mathcal{D}, \mathcal{C})$[1] where $\mathcal{V}$ is the set of n design variables $\{V_1, V_2, \ldots, V_n\}$, $\mathcal{D}$ is the set of n domains[2] of each variable $\{D_1, D_2, \ldots, D_n\}$ and $\mathcal{C}$ is the set of p relations among variables, called constraints $\{C_1, C_2, \ldots, C_p\}$ [13].

There is evidence in the literature that constraint-based methods are widely used to model Engineering Design problems [1, 7–9, 17]. The key point is that this knowledge, issued from expert's experience, should be properly elicitated, structured and stored in order to be reused to enrich future product developments. The issue increases by coordinating efforts in distributed and heterogeneous knowledge modelling derived from CE approaches. Other proposal have been made to collaborative model the design process in form of CSP and an algorithm that considers multiple actors was proposed to as many integrate product lifecycle constraint will be possible [21].

In previous works a knowledge modelling structure was proposed to represent and store CSP variables in for knowledge design [25]. The stored knowledge was used by a Multi-Agent System to make a pre-validation of the variables in the model by comparing them using the attributes define. The agents reuse the knowledge captured to suggest similar variables to the experts in order to accelerate the definition process of new components in the model.

In spite all the proposals made to represent engineering design knowledge using CSP or constraint programming, there is any proposal using a standard representation that provides the means to transport, reuse and communicate the knowledge.

[1]From the triple $\langle X, D, C \rangle$ definition of Constraint Satisfaction Problem (CSP) theory that defines CSP as mathematical problems composed by as a set of objects whose state must satisfy a number of constraints.

[2]Set of possible values that a variable V_i can take.

2.2. Set of Experience Knowledge Structure

Knowledge have played a differentiator role in the human history. Nowadays, knowledge is probably the most important asset of civilizations and companies and its value is appreciated for nation's in military and economic aspects as well as for companies for creating competitive advantage. This fact has impulsed the world to turn into a knowledge society, making necessary the development of technologies to manage and control all forms of knowledge because this technology can be consider the platform the succeed.

Due to this, Knowledge-based Systems were created based on inferences mechanism and formalisms that were not more than a set of rules. This rules were elicitated from the expert by interviews and by the understanding of the activity to be modelled [19]. Then, by a transfer process, this knowledge was implemented in a production rules that were executed by an Interpreter. This approaches were not successful due to representing knowledge by production rules did not support adequate representation of different types of knowledge [27].

After this attempt of knowledge representation with rules, a new brand of Artificial Intelligence emerged, Knowledge Engineering (KE). Feigenbaum defines KE as "an engineering discipline that involves integrating knowledge into computer systems in order to solve complex problems normally requiring a high level of human expertise" [11]. Since the beginning, the engineers working in these topics developed principles, methods and techniques to identification, capturing and representation of knowledge [4, 14, 18, 23] among others.

Other technologies like Knowledge Management Systems (KMS), Data Mining, among others, were also developed for different proposals and for working with different kind of knowledge. Each of them work with decision-making in some way but they do not have structure knowledge of the formal decision events they participate in. Set of Experience Knowledge Structure (SOEKS) has been developed to keep formal decision events in an explicit way [6]. SOEKS is a model that structure knowledge based on four basic components that surround decision-making events: *variables V, functions F, constraints C, and rules R*.

One of the first ways to represent knowledge was variables, this means, using an attribute-value relationship. SOEKS uses variables as a root for its structure because variables are the source of the other three components. Functions, describes associations between a

dependent variable and a set of input variables and are used in SOEKS for established links among variables and to construct optimal states or representing multi-objective goals when taking a decision. The third component, *constraint*, are another way of relationship among the variables but they have different purpose than the functions. Constraints in SOEKS acts a limitation of the possibilities, restricting the solutions in a decision problem. Finally, the rules, are also relationships between the variables expressed in the form *IF-THEN-ELSE*.

The combination of the four elements of SOEKS gives uniqueness to each result, making possible to represent formal decision events. The possibility offer by SOEKS to be represented using multi-platform technologies like `XML` makes this approach excellent for exchanging experiences across multiple platforms and make it appropriate to exchange and reuse design knowledge in a concurrent environment.

3. SOEKS based collaborative CSP definition

SOEKS was proposed as a knowledge representation to keep decision events in an explicit way [6]. As mentioned above, SOEKS is composed by four components: *Variables V, Functions F, Constraints C, and Rules R*. An experience can be represented by a combination of all these components, likewise, an experience might be represented uniquely by a set of values of one of the components, *e.g.* a set of six variables *V* represents a movie in a Web Data Mining Application [22].

In previous work a knowledge representation structure was defined for representing engineering design knowledge modelling, by a tuple *Variables V, Domains D and Constraints C* [25]. The proposal was based on the idea that a design problem can be represented as a CSP model in order to be treated by an inference engine to obtain a set of possible solutions. The definition of each element of the model, is an explicit representation of the knowledge from the expert that defined it. This approach includes a definition of the variables' attributes regarding, not only the characteristics that defined the variable itself, but also includes some attributes that are necessary to define a CSP model in CON'FLEX[3] syntax and in collaborative and concurrent environments. The definition of constraints is based in how CON'FLEX categorizes it.

[3] A solver of Constraint Satisfaction Problems. Source: http://carlit.toulouse.inra.fr/conflex/ Last accessed: February 22nd 2013.

Both approaches, SOEKS and CSP, agree on the definition of variables and constraints as elements of knowledge but a deeper analysis of the Variables' attributes and Constraints types show that these approaches have some differences that difficult the direct representation of CSP knowledge into SOEKS structure. This section presents the differences between both approaches and the modifications that had to be made to be able to use a generic knowledge representation for supporting the concurrent definition of CSP models for product design.

3.1. Variable representation

Conceptually, SOEKS interprets variables as the representation of the current state of the external environment and which can be modified by an action that changes the value of those variables and therefore, producing a new state [5, p. 42]. Thus, variables are defined as an object with eleven attributes; six of them are mandatory while the other five are optional see [5, p. 83]. The XML representation of a variable can be seen in Figure 1. The attributes that describes the variable are: `(<var_name>)` for the name of the variable; `(<var_type>)`, the type of variable: *Numerical* or *Categorical*; the cause value `(<var_cvalue>)`, the value of the variable in the first state, before being optimized; `(<var_evalue>)` representing the effect value, which is the value of the variable after being optimized; `(<unit>)`, what represents the values; a Boolean `(<internal>)` to represent if the value is internal or external. In addition of these attributes, variables also include optional attributes necessary in the processes of similarity, uncertainly, impreciseness, or incompleteness measures. Those attributes are `(<weight>)`, for measuring the weight of the variables; `(<priority>)`, to indicate the priority; `(<l_range>)` lower range and `(<u_range>)`, upper range, to define the values that the variables can take; and `(<categories>)`, that can be more than one and are used when the variable is *categorical*.

According to the CON'FLEX syntax, a variable can be of three types: *Integer*, *Real* and *Symbolic*. According to the type of variable, there are different attributes associated with it. Thus, *Symbolic* variable has the following attributes: `Name`, which has to be unique; `Domain` of the variable, which is a list with the symbolic values that the variable can take; and an optional value, the `Degree of membership`, to indicate the degree of membership of the value to the inference engine, by default this value is 1.

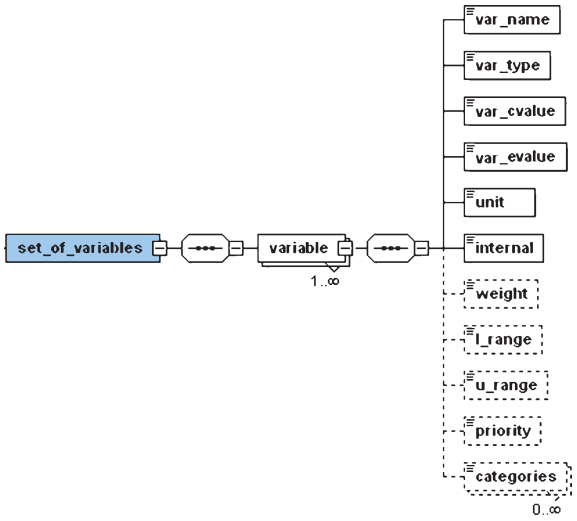


Fig. 1. Tree of variables of a SOEKS-XML configuration [5].

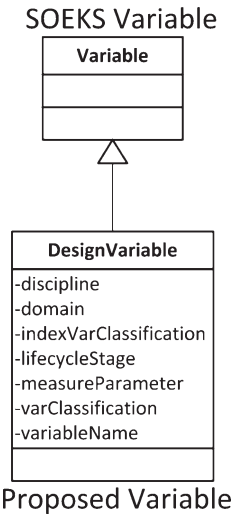


Fig. 2. Class diagram for design variable.

The *Integer* variable also has a `Name`; that has to be unique; an integer `Domain` that can be of three types: a list of integer values; a range of values limited by the upper and lower numbers; and a combination them. An *Integer* variable also has a `Degree of membership` (between 0 and 1) that can be assigned to each of the values of the domain.

In CSP *Real* variables are declared using three attributes: `Name`; `Step`, which represents the increasing of the variable per iteration (0.1 by default); and finally `Domain`, for which a real variable can be defined in four ways: As a closed interval `[<a>, <b>]` and the support of the fuzzy interval of height 1; As a closed interval `[<a>, <b>, (<d>)]` with the interval `[a, b]` and the height *d* between 0 and 1 for the last *alpha-cut*; As a closed interval `[<S1>, <n1>, <n2>, <s2>]` with the number `[s1, s2]` that represents the support, and the core of the fuzzy interval `[n1, n2]` with default height 1; and a closed interval `[<s1>,<n1>,<n2>,<s2>, (<d>)]` with support `[s1, s2]`, the highest alpha-cut `[n1, n2]` and the height *d*, between 0 and 1.

In spite of the similarities that a first glance may exist between the variables in CON'FLEX and the variables in SOEKS, there are some major differences that prevent to directly use SOEKS for representing CON'FLEX's variables. The main difference is the number of attributes needed in CON'FLEX to represent a variable and the number of attributes that SOEKS has. In addition, some of the attributes that apparently are common to both proposals are missing some conceptual aspects necessary to represents all the attributes of a variable for CON'FLEX.

Therefore an extension of the variable is proposed to include the attributes that are not in the definition of Variable in SOEKS. The new variable is called Design Variable *(DV)* and is an Extension of the class Variable proposed in [5, p. 42-46]. The new class has the attributes shown in the Fig. 2.

`Discipline` and `lifecycleStage` are attributes needed to identify variables while experts are defining the CSP model in a collaborative and concurrent environment. `Discipline` attribute refers to the discipline in which the variable is used, i.e. *Shopping, Geometric, etc*. and `lifecycleStage` refers to which of the product lifecycle stages the variable is used. `Domain` has the values that the variable can take and its format depends on the type of the variable. The domain logic is validated in the code in order to make easily the definition of variables. The `indexVarClassification` allows the classification of the variable in four categories: *Morphological, Criterion, Physical, and Technical*.

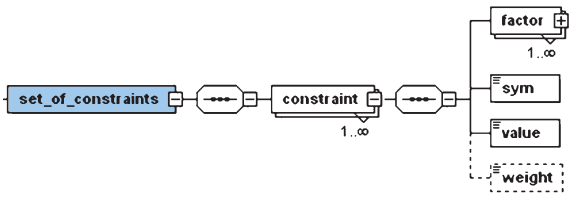


Fig. 3. Tree of constraint of a SOEKS-XML configuration [5].

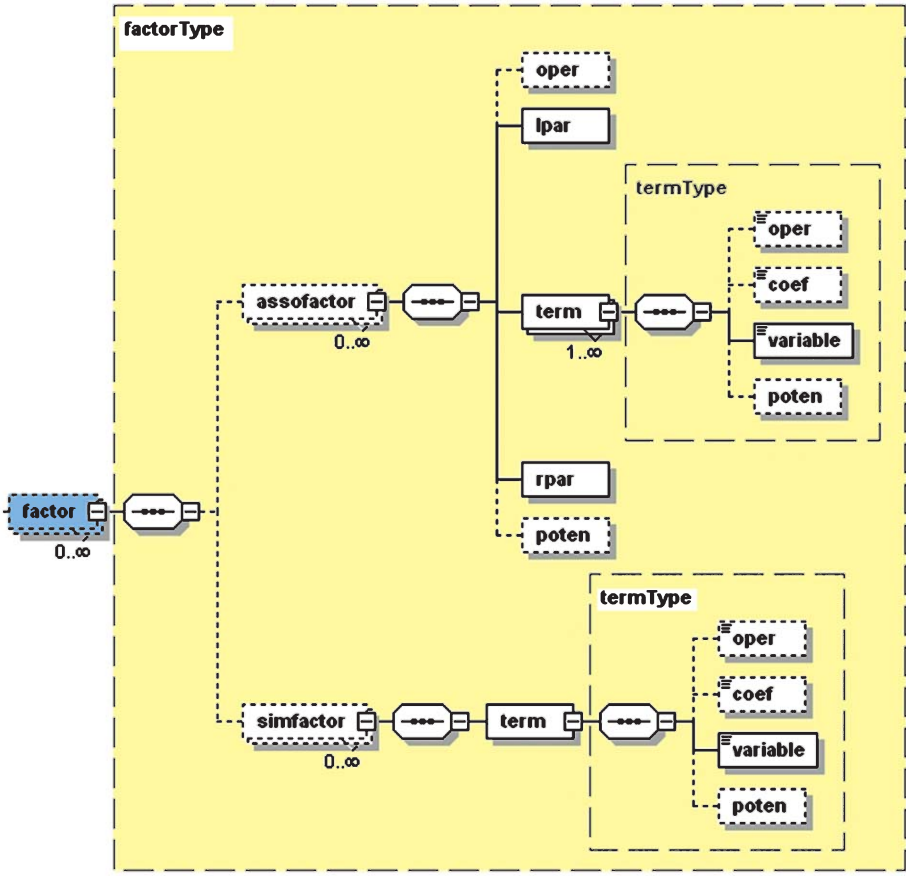


Fig. 4. Tree of factor of the SOEKS-XML configuration [5].

The attribute `measureParameter` is used to represent the measure parameter of the variable, that can be: *length, mass, etc*. And finally, a *DV* has two names, the one in the father class, `varName`, which it is used to represent the variable in a technical way. This name specifies the type of variable and a consecutive number to identify it. The second name, `variableName` which is used to present the variable to the experts with a name easily to remember and that can be associated with the function of the variable in the model.

By extending SOEKS's variable it is possible to have a representation for the basic design knowledge. This representation, is an extended version that not only includes the normal attributes of the variables but also the attributes of the design variable. With these new attributes, it will be feasible to make new inferences about the CSP model and even about the experiences of the experts during the definition of the models.

3.2. Constraint representation

An initial look to the SOEKS and its structures suggests that *Constraints* for a CSP model can be represented using the *Constraint* structure of SOEKS. Constraints in SOEKS are expressed by four elements: *factor, sym, value*, and optional value: *weight* (see Fig. 3).

In a constraint expression the element *factor* have to be at least one time and it is expressed as shown in Fig. 4.

Factors are constituted by two types of factors: `(<assoactor>)` and `(<simfactor>)`. The *assofactor* is a structure to represents associated factors and contains the following elements: parenthesis `(<lpar>, <rpar>)`; two optional elements: the operator `(<oper>)` and the potency `(<poten>)`; and terms `(<term>)`. *term* is an structure itself and contains: `(<oper>)`, coefficient `(<coef>)`, variable `(<variable>)` and `(<poten>)`. *simfactor*

represents single factors and therefore it only includes *terms*.

The structure proposed in SOEKS for *Constraint* its a good foundation to represent CSP's constraint but, due to the syntax and the conceptual definition of this element in CON'FLEX, not all of the attributes and aspects are cover with this approach. Therefore, it is necessary to use not only the *Constraint* structure but other structures defined in SOEKS to be able to fully represent all types of *Constraints* and their attributes. Furthermore, it is also necessary to define a conceptual way to integrate the new form to represent constraints into a single SOEKS. This allow the representation of design experiences of the experts and also, will permit making inferences with this knowledge. As a result, the three principals constraints in CON'FLEX: *Extension, Intention and Conditional*, where represented following the syntax of CON'FLEX but based on different SOEKS's structures.

3.2.1. Extension constraints

In CON'FLEX *Extension constraint* are used to list explicitly and exhaustively all the combination of possible values of set of variables. This kind of constraint is only defined when the expert is uncertain about some aspects of the design or when there are variables that are related.

An *extension constraint* can be seen as collection of rules where the condition is the relationship between variables, and consequence is the degree of compatibility of the tuple of that relationship (between 0 - 1). According to this, the best SOEKS's structure to represent each relationship is *rule* (see [5, p. 87-88]).

To represent the *extension constraint* a new class was defined, `ExtensionConstraint`. The class extends the `RuleSet` Class, that is a SOEKS's structure which has as attribute a `Vector` of `Rule`. Each rule is composed by two mandatory elements: `(<join>)` to represents the conditions; and the second one, `(<consequence>)`, that corresponds to the consequence. The other two are optional elements necessaries to make inferences process in SOEKS: `(<confidence>)`, to offer certainty on action of the rule, and `(<weight>)`. In turn, each of the mandatory elements has their own elements, `(<join>)` has `(<jnt>)` to allow the occurrence of more than one `(<join>)` in the rule by connecting them with a logic connector (i.e. AND / OR); and `(<condition>)`, that is composed by `(<factor>)`; `(<sym>)`; `(<value>)`; `(<variable>)`; and `(<weight>)` to represent the importance of the rule among the set of rules it belongs. Consequence `(<consequence>)`, has four more elements: `(<variable>)`; `(<sym>)`; `(<value>)`; and `(<variable>)`.

Additionally it is necessary to define three more attributes in the Class `ExtensionConstraint` to be able to translate it into a CSP model. These attributes are: `name`, which is the name of the *Extension Constraint* to be presented to the user; the representation symbol of the constraint is `representationSymbol` and the priory (Importance) of the constraint in the model, by default 1.

To save an *extension constraint* into a SOEKS object, each rule of the constraint is store as a `rule` and its name have to follow the next structure:

```
constraintName-repSymbol-i.
```

Where `constraintName` is the name given to the constraint, the representation symbol of the variable is the model is `repSymbol`, and `i` is the position of the rule in the `Vector` of rules.

With this convention to represent the `name`, it is possible to go from *Extension Constraint* objects to *SOEKS*'s objects and from *SOEKS* to *Extension Constraint* without loosing information or the capabilities of SOEKS. Also, it is possible to represent *Extension constraint* as experience in SOEKS.

3.2.2. Intention constraints

Intention constraints in CON'FLEX are relationships between the variables declared in the model. In CON'FLEX an *intention constraint* has to be define as:

```
<name> [(<priority>)] , <equation>
```

Where `name` is the representation symbol used in the model to identify the constraint. By convention this name is usually defined as the initial letter of the type of constraint together with a consecutive number separated by a dash *i.e.* CI-1. `priority` defines the importance of the constraint (1 per default) and `equation` which is the functional relationship following the form: `<exp><binop><exp>` where: `<binop>` is the binary operator in the equation that can be: $=, \leq, <, \geq, >, \neq$. The element `<exp>` includes any type of variable (previously defined in the model), arithmetic operators: $+, -, *, /, x^y$ (integer power), $**$(real power in $\Re$) and $\sqrt{x}$. *i.e.* $x^2 - 3x + 2/y \leq -0.5$.

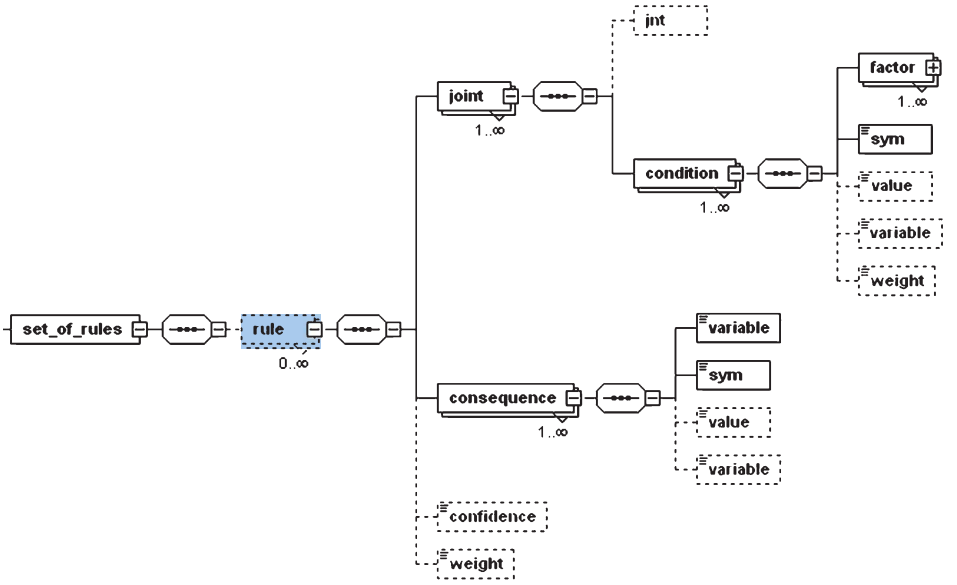


Fig. 5. Tree of rules of the SOEKS-XML configuration [5].

To be able to represent an *intention constraint* in SOEKS the Class `IntentionConstraint` was defined. This Class extends the `Constraint` Class (see [5, p. 87-89]) due to it has a similar structure than the defined in CON'FLEX. The left side of the `equation` is represented in SOEKS by the element `factor`, `<binop>` is `<sym>`, the `weight` attribute is the equivalent to `priority`, and the expression of right side of the equation is the element `value`. To be able to make this transformation and represent the experience of defining an *Intention Constraint* in a SOEKS, it is necessary to equalize the `equation` to zero. By doing this, all of the variables will be in the left side of the `equation` and the right side will have the value. Another reason to extends the `Constraint` Class was the missing of one attribute needed it the definition of CSP models, the `representationSymbol`.

With this approach it is possible to include an *intention constraint* as an element of the experience represented in SOEKS and reuse by making inferences based on those experiences, tutoring experts by giving them clues and help while defining a new CSP model, among others.

3.2.3. Conditional constraints

A *Conditional Constraint* has two sets of constraints: the *premise* and the *conclusion*. Both are composite by *intention* and *extension* constraints but for experimental purposes all the *extensions* will be transform into *intention* constraints. The syntax in CON'FLEX to represent this kind of constraint is:

```
\cc : <name>
     \if
        ...
        <PC_i>
        ...
     \then
        ...
        <CC_j>
        ...
     ;
```

Where `<name>` is the name of the *Conditional Constraint* and by convention has the form: `CC#`, with # representing a consecutive number that differentiate one constraint from another. The `<PC_i>` in the `\if` clause is the element `i` of the set of *premises*, in the other hand `<CC_j>` in the `\else` clause, is the `j` element of the *conclusion* constraints.

To be able to use SOEKS' structures it was necessary to modify the SOEKS source code and changed the `Consequence` Class. Originally a `rule` could be represented in XML as seen in Fig. 5.

Due to *conclusion* is a set of *Intention Constraints*, an attribute of the Class had to be changed. `lhsVariable`, which is `Variable` type, was changed by a `Factor` type attribute to able to include

intention constraints into the *conclusion*. With this change, it is possible to presents *Conditional Constraints* in SOEKS and therefore, create a SOEKS object with the experience of defining the Conditional Constraint by adding rules that can have conclusions composite by *intention constraints*.

4. Case study

This section describes a simplification of the engineering design process used by a company specialized in designing and building polymerization plants. The polymerization process is classified as part of the AEC

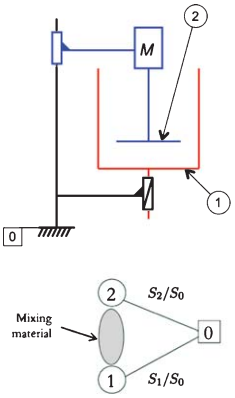
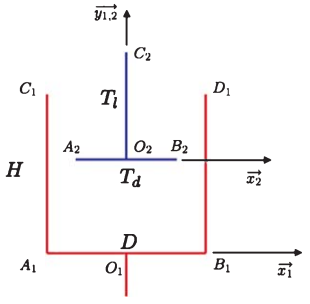


Fig. 6. Mixer kinematic diagram.

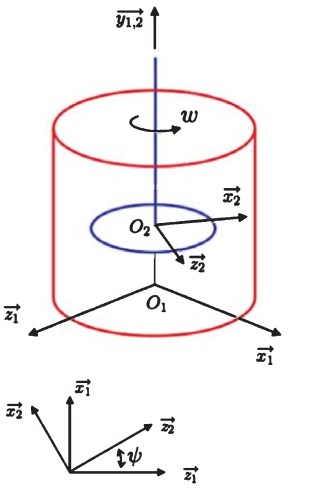


$$S_1 \left\{ \begin{array}{rcl} \overrightarrow{A_1B_1} = \overrightarrow{C_1D_1} & = & D\,\overrightarrow{x_1} \\ \overrightarrow{B_1D_1} & = & H\,\overrightarrow{y_1} \end{array} \right.$$

$$S_2 \left\{ \begin{array}{rcl} \overrightarrow{A_2B_2} & = & T_d\,\overrightarrow{x_2} \\ \overrightarrow{O_2C_2} & = & T_l\,\overrightarrow{y_2} \end{array} \right.$$

$$S_1/S_2 \left\{ \begin{array}{rcl} \overrightarrow{O_2O_1} & > & 0 \\ \overrightarrow{A_2B_2} & < & \overrightarrow{A_1B_1} \end{array} \right.$$

Fig. 7. Mixer S_1/S_2 morphology analysis.



$$\mathcal{T} \left\{ \begin{array}{rcl} \overrightarrow{\Omega_2^1} & = & \dot{\psi}\,\overrightarrow{y_2} = w \\ \overrightarrow{V_2^1(O_2)} & = & 0 \end{array} \right\}$$

Fig. 8. Mixer S_1/S_2 kinematic analysis.

(Architecture, Engineering and Construction) sector, which are composed of several subsystems such as piping, civil engineering, specialized equipment (pumps, mixers, etc.) design and selection. The company defines together with the client a set of functional requirements for a plant construction. Each plant functionality usually changes depending on different factors such as: type of polymer to mix, way of mixing, chemical reactions experimented during the mixing process, temperature reached by the mix, security regulations, pressure supported by the reactors, minimum and maximum volume of mix, among others. Most of the expended time during the design of the whole plant is dedicated to analyse and to evaluate which is the best design for the pre-reactors and reactors.

The object of study in this article is one of the most common components: The mixer. The engineering problem design problem is the design and dimensioning of an industrial mixer. The literature has presented a simplification of this analysis as the case presented by Gelle [12]. This generalization is combined with some factors used by the company during their design process. This case enables to validate that the changes proposed to SOEKS and the conceptual interpretation given to some of its structures allows to represent design experiences in terms of *variables, constraints* and *rules* regardless if *rules* and *constraints* link discrete, continuous or symbolic *variables*.

As part of the engineering process to design industrial mixers, some general concepts will be introduced. There are two types of components: the "mixer", which describes the product itself, and "the type of mixture" which determines the requirements according to the type of polymer to mix. Additionally, industrial mixers can be classified into three types: reactor, storage or conventional[4]. Similarly, depending on the combination of phases (solid, liquid or gas) from the mixed product(s), the type of mixture can be defined as: suspension, mixing, dispersion or moving.

It has been presented so far the general architecture of the knowledge model in terms of variables. Implicitly, the possible values that a variable can take were discussed. It is therefore introduced the concept of "domain", which is fundamental to a CSP model definition. For the type of mixture case, the variable was named `VS1_MT_subtype` and its corresponding domain will be as Equation 1.

$$VS1\_MT\_subtype = \{suspension, mixing, dispersion, moving\} \quad (1)$$

In an equivalent manner, the engineering process is developed to identify more variables and relations among them (to find constraints). One of the main types of constraints are those from the morphological analysis, as well as from product behaviour (e.g. dynamics).

According to Figs. 6, 7 and 8, each component of the mixer, represented as a rigid body in the kinematic diagram, is analysed in order to bring out the first geometric constraints.

From these analysis, it is possible to define more variables. Some examples are presented in Equation 2. The set of domains for these variables are defined as real values.

$$\begin{cases} D = < \texttt{VR3\_V\_d} > & \text{Vessel diameter} \\ H = < \texttt{VR4\_V\_h} > & \text{Vessel height} \\ T_d = < \texttt{VR6\_I\_d} > & \text{Impeller diameter} \\ w = < \texttt{VR7\_I\_rps} > & \text{Impeller angular speed} \end{cases} \quad (2)$$

In summary, the model has 15 variables between symbolic and real, that are related by 7 constraints. The mixer was modelled in 4 components and variables were grouped in those four elements of the mixer: the *Mix Tasks*; the *Vessel* configuration: Volume, Diameter and Height; the types of impellers and their physical parameters as well as Impeller's *RPS* and

[4]This particular case will be simplified to the conventional mixer.

CSP

```
\vs : VS1_MT_subtype suspension
                     dispersion
                     entrainment
                     blending ;
```

SOEKS

```
<variable>
    <var_name>VS1</var_name>
    <var_type>symbolic</var_type>
    <var_cvalue>suspension</var_cvalue>
    <var_evalue>suspension</var_cvalue>
    <unit>mix type</unit>
    <internal>true</internal>
    <weight>1</weight>
    <l_range>suspension</l_range>
    <r_range>blending</r_range>
    <prionity>1</prionity>
    <categories>Suspension</categories>
    <categories>dispersion</categories>
    <categories>entrainment</categories>
    <categories>blending</categories>
    <variable_name>MT_subtype</variable_name>
    <var_classification>Technical</var_classification>
    <measure_parameter>symbolic</measure_parameter>
    <lifecicle_state>conception</lifecicle_state>
    <discipline>geometric<discipline>
</variable>
```

Fig. 9. Transformation of symbolic variable from CSP to SOEKS structure.

CSP

```
\vs : VS3_V_subtype cylindrical ;
\vr : VR2_V_v 0.1 [0.1,9] ;
\vr : VR3_V_d 0.1 [0.2,2] ;
\vr : VR4_V_h 0.1 [0.1,2.6] ;
```

SOEKS

```
<variable>
    <var_name>VR2</var_name>
    <var_type>real</var_type>
    <var_cvalue>0.1</var_cvalue>
    <var_evalue>0.1</var_cvalue>
    <unit>cubic meter</unit>
    <internal>true</internal>
    <weight>1</weight>
    <l_range>0.1</l_range>
    <r_range>9</r_range>
    <prionity>1</prionity>
    <variable_name>V_v</variable_name>
    <var_classification>physical</var_classification>
    <measure_parameter>volume</measure_parameter>
    <lifecicle_state>conception</lifecicle_state>
    <discipline>geometry<discipline>
    <domain>0.1[0.1,9]</domain>
</variable>
```

Fig. 10. Transformation of real variable from CSP to SOEKS.

power; finally there is a variable with the power of the Engine.

Similarly, technical experiences can be modelled as mathematical expressions to obtain constraints and reused them in similar projects. A clear example for this can be the Volume of the Vessel as well as the power needed by the motor to activate the mixer. Consequently, the knowledge of the experts are related by the definition of seven *Constraints*. The first constraint in the model limits the *vessel volume* by relating the vessel's height and diameter with the constraint π. The second constraint links the type of Impeller with its angle; the next four constraints relate the Impeller type

CSP
```
\ci : CI1 , VR2_V_v = ((1/4)* PI * VR3_V_d^2 * VR4_V_h) ;
```

SOEKS
```
<constraint>
    <name>CI1</name>
    <factor>
        <simfactor>
            <term>
                <variable>VR2_V_v<variable>
            </term>
        </simfactor>
        <assofactor>
            <oper> - </oper>
            <l_par>(<l_par>
            <term>
                <variable>1/4<variable>
            </term>
            <term>
                <oper>*</oper>
                <variable>PI<variable>
            </term>
            <term>
                <oper>*</oper>
                <variable>VR3_V_d<variable>
                <poten>2</poten>
            </term>
            <term>
                <oper>*</oper>
                <variable>VR4_V_h<variable>
            </term>
            <l_par>)<l_par>
        </assofactor>
    </factor>
    <sym>=</sym>
    <value>0</value>
    <weight>1</weight>
</constraint>
```

Fig. 11. Transformation of intention constraint from CSP to SOEKS.

with its diameter and a power constant, which is used in the last constraint to obtain the Power.

Based on the previous details, the design knowledge was modelled under the CSP approach and stored under the modified SOEKS structure, in order to be stored and reused in further applications. Figures 9, 10 and 11 shows an example of a symbolic variable, real variable and a intention constraint, respectively.

5. Conclusions

In this paper the importance of Constraint Satisfaction Problems (CSP) and Constraint programming for linking up knowledge to find better solutions for a design problem was presented. Additionally, the definition and structure of CON'FLEX was introduced. In the article there was also presented the importance of having a well defined knowledge representation structure in different Artificial Intelligence topics and due to that, a brief introduction to SOEKS was given. SOEKS has been developed to keep formal decisions events in an explicit way by the definition of four components: *Variables, Functions, Constraints and Rules*. This components can be combined to produce unique results and also can be represented using `XML`. A comparison between the attributes of CON'FLEX and SOEKS was done and based on the differences, an extended representation structure was proposed. The proposal modifies some structures of SOEKS and makes different interpretations of some of its attributes to be able to reap the advantages of represent Product Design knowledge as a SOEKS. Finally a case study is presented in order to validate the proposed approach.

Having the design knowledge represented in a technology that allows the transportation and re-utilization of knowledge across different applications, lets us think about new applications of the represented knowledge. Further research have to be done to create inferences systems for the design experiences represented and stored in SOEKS.

Acknowledgements

This work was supported by Universidad EAFIT. The authors appreciate the partial financial support from the Medellin City (Colombia) through the "Enlazamundos" program that was granted for supporting an exchange period in Australia to develop part of this project. We are also grateful to the University of Newcastle, Australia for opening its doors to make possible this collaborative work.

References

[1] A.J. Qureshi, J.-Y. Dantan, J. Bruyere and R. Bigot, Set based robust design of mechanical systems using the quantifier constraint satisfaction algorithm, *Engineering Applications of Artificial Intelligence* **23**(7) (2010), 1173–1186.

[2] A. Thornton and A. Johnson, CADET: A software support tool for constraint processes in embodiment design, *Research in Engineering Design* **8** (1996), 1–13.

[3] A.C. Thornton, The use of constraint-based design knowledge to improve the search for feasible designs, *Engineering Applications of Artificial Intelligence* **9**(4) (1996), 393–402.

[4] B.J. Wielinga, A.T. Schreiber and J.A. Breuker, KADS: A modelling approach to knowledge engineering, *Knowledge acquisition* **4**(1) (1992), 5–53.

[5] C. Maldonado Sanin, *Smart knowledge management system, PhD Thesis, Faculty of Engineering and Built Environment - School of Mechanical Engineering*, The University of Newcastle, Newcastle, Australia, 2007.

[6] C. Sanin and E. Szczerbicki, Experience-based knowledge representation: SOEKS, *Cybernetics and Systems: An International Journal* **40**(2) (2009), 99–122.

[7] D. Scaravetti, J.-P. Nadeau, J. Pailhes and P. Sebastian, Structuring of embodiment design problem based on the product lifecycle, *International Journal of Product Development* **2**(172) (2005), 47–70.

[8] D. Scaravetti, J. Pailhes, J.-P. Nadeau and P. Sebastian, Aided decision-making for an embodiment design problem, in: A. Bramley, D. Bris- saud, D. Coutellier, C. McMahon (Eds.), *Advances in Integrated Design and Manufacturing in Mechanical Engineering*, Springer Netherlands, (2005), pp. 159–172.

[9] D. Yang and M. Dong, A constraint satisfaction approach to resolving product configuration conflicts, *Advanced Engineering Informatics* **26**(3) (2012), 592–602.

[10] D. Yang, M. Dong and X.-K. Chang, A dynamic constraint satisfaction approach for configuring structural products under mass customization, *Engineering Applications of Artificial Intelligence* **25**(8) (2012), 1723–1737.

[11] E.A. Feigenbaum and P. McCorduck, *The fifth generation: Artificial intelligence and Japan's computer challenge to the world*, Addison-Wesley Longman Publishing Co., Inc., 1983.

[12] E. Gelle, On the generation of locally consistent solution spaces in mixed dynamic constraint problems, Ph. D. Thesis, *Ecole Polytechnique Federale de Lausanne*, Switzerland, 1998.

[13] E. Tsang, *Foundations of Constraint Satisfaction*, Vol. 289, Academic Pr, 1993.

[14] G. Schreiber, B. Wielinga, R. de Hoog, H. Akkermans and W. Van de Velde, CommonKADS: A comprehensive methodology for KBS development, *IEEE expert* **9**(6) (1994), 28–37.

[15] J.A. Harding, *A knowledge representation model to support concurrent engineering team working, Phd thesis, mechanical and manufacturing engineering*, Loughborough University, 1996.

[16] J.G. McGuire, D.R. Kuokka, J.C. Weber, J.M. Tenenbaum, T.R. Gruber and G.R. Olsen, SHADE: Technology for knowledge-based collaborative engineering, *Concurrent Engineering* **1**(3) (1993), 137–146.

[17] M. Aldanondo, E. Vareilles, K. HadjHamou and P. Gaborit, Aiding design with constraints: An extension of quad trees in order to deal with piecewise functions, *International Journal of Computer Integrated Manufacturing* **21**(4) (2008), 353–365.

[18] M. Ester, H.-P. Kriegel and X. Xu, *Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification,* in: Advances in Spatial Databases, Springer, (1995), pp. 67–82.

[19] M. Musen, *An Overview of Knowledge Acquisition,* in: J.-M. David, J.-P. Krivine, R. Simmons (Eds.), *Second Generation Expert Systems*, Springer Berlin Heidelberg, (1993), pp. 405–427.

[20] M.R. Cutkosky, R.S. Engelmore, R.E. Fikes, M.R. Genesereth, T.R. Gruber, W.S. Mark, J.M. Tenenbaum and J.C. Weber, *PACT: An experiment in integrating concurrent engineering systems, Computer* **26**(1) (1993), 28–37.

[21] P.-A. Yvars, A CSP approach for the network of product lifecycle constraints consistency in a collaborative design context, *Engineering Applications of Artificial Intelligence* **22**(6) (2009), 961–970.

[22] P. Wang, C. Sanin and E. Szczerbicki, Application of decisional DNA in web data mining, *Knowledge-Based and Intelligent Information and Engineering Systems* **6882** (2011), 631–639.

[23] Q.Y. Fu, Y.P. Chui and M.G. Helander, Knowledge identification and management in product design, *Journal of Knowledge Management* **10**(6) (2006), 50–63.

[24] R. Davis, H. Shrobe and P. Szolovits, What is a knowledge representation? *AI magazine* **14**(1) (1993), 17.

[25] R. Mejía-Gutierrez, A. Cálad-Álvarez and S. Ruiz-Arenas, *A Multi-Agent Approach for Engineering Design Knowledge Modelling,* in: Knowledge-Based and Intelligent Information and Engineering Systems, Part II, Vol. 6882 of Lecture Notes in Computer Science, A. Koenig, A. Dengel, K. Hinkelmann, K. Kise, R.J. Howlett, L.C. Jain, (Eds.), Kaiserslautern, Germany, (2011), pp. 601–610.

[26] T. Luz, E. Loup-Escande, H. Christofol and S. Richir, *The collaborative product design and help to decision making: Interactive mind-mapping*, in: A. Bernard (Ed.), Global Product Development, Springer Berlin Heidelberg, (2011), pp. 237–244.

[27] W.J. Clancey, The epistemology of a rule-based expert system a framework for explanation, *Artificial Intelligence* **20**(3) (1983), 215–251.