what-when-how

In Depth Tutorials and Information

Configuration (Artificial Intelligence)

INTRODUCTION

Configuring means selecting and bringing together a set of given components to produce an aggregate (or a set of aggregates) satisfying some requirements. All the component types are predefined and no new component type may be created during the configuration process.

The result of the configuration can be physical objects (such as cars or elevators), non-physical entities (such as compound services or processes) or heterogeneous wholes made of both physical and non-physical parts (such as computer systems with their hardware and software components).

The configuration process has to take into consideration both endogenous and exogenous constraints: the former pertain to the type of the assembled object(s) (therefore they hold for all the individuals of that type) and mainly come from the interactions among components, whereas the latter usually represent requirements that the final aggregate(s) should satisfy. All these constraints can be very complex and make the manual solution of configuration problems a very hard task in many cases.

The complexity of configuration and its relevance in several application domains have stimulated the interest in its automation. Since the beginning, Artificial Intelligence has provided various effective techniques to achieve this goal. One of the first configurators was also one of the first commercially successful expert systems: a production rule-based system called R1 (McDermott, 1982, 1993). R1 was developed in the early Eighties to configure VAX computer systems, and it has been used for several years by Digital Equipment Corporation.

Since then, configuration has gained importance both in industry and in marketing, also due to both the support that it offers to the mass customization business strategy and the new commercial opportunities provided by the Web. Configuration is currently an important application field for many Artificial Intelligence techniques and it is still posing many interesting problems to scientific research.

BACKGROUND

The increasing complexity and size of configurable products made it clear that production-rule-based configurators such as R1 are not effective, particularly in the phase of maintenance of knowledge bases. In fact, changing a rule may require, as a side effect, changing several other rules and so on, and, actually, for some products, the component library may change frequently.

To partly address this problem, in current configurator systems, domain knowledge and control knowledge for problem solving are separate. The domain knowledge is represented in a declarative language, and the control knowledge (i.e., inferential mechanisms) is general (i.e., not depending on the particular problem to be solved). This is a common approach in modern knowledge-based systems. A configurator is based on an explicit representation of the general model of the configurable entities, which implicitly represents all the valid product individuals. The reasoning mechanisms implement the control knowledge and they use the domain knowledge to draw inferences and to compute configurations.

Regarding domain knowledge, there is a general agreement about what the concepts to represent are. In (Soininen, Tiihonen, Mannisto & Sulonen, 1998) the authors introduce a widely accepted conceptualization for configuration problems. This conceptualization includes the concepts of model the compositional structure of PCs (e.g., each PC has one or two monitors, a motherboard, etc.). Each component of a PC can be either of a basic (non configurable) type (e.g., the monitor) or of an aggregate (possibly configurable) type (e.g., the motherboard). Some relevant taxonomic relations are reported (e.g., the hard disks are either SCSI or EIDE). The basic components can be connected through ports (only few ports are reported): each port connects with at most one other port; for some ports the connection is optional (e.g., for eide_port), for others it is mandatory (e.g., for device_eide_port). Some attributes (e.g., the price) describe the components. A set of constraints model the interactions among the components: e.g., the third constraint specifies that hard disks must provide enough space, which is a resource consumed by operating systems.

• **components,** which are the constituents of configurations;

• **parts to** describe the compositional structure;

• **ports to** model connections and compatibilities between components;

• **resources that** are produced, used or consumed by components;

• **functions to** represent functionalities;

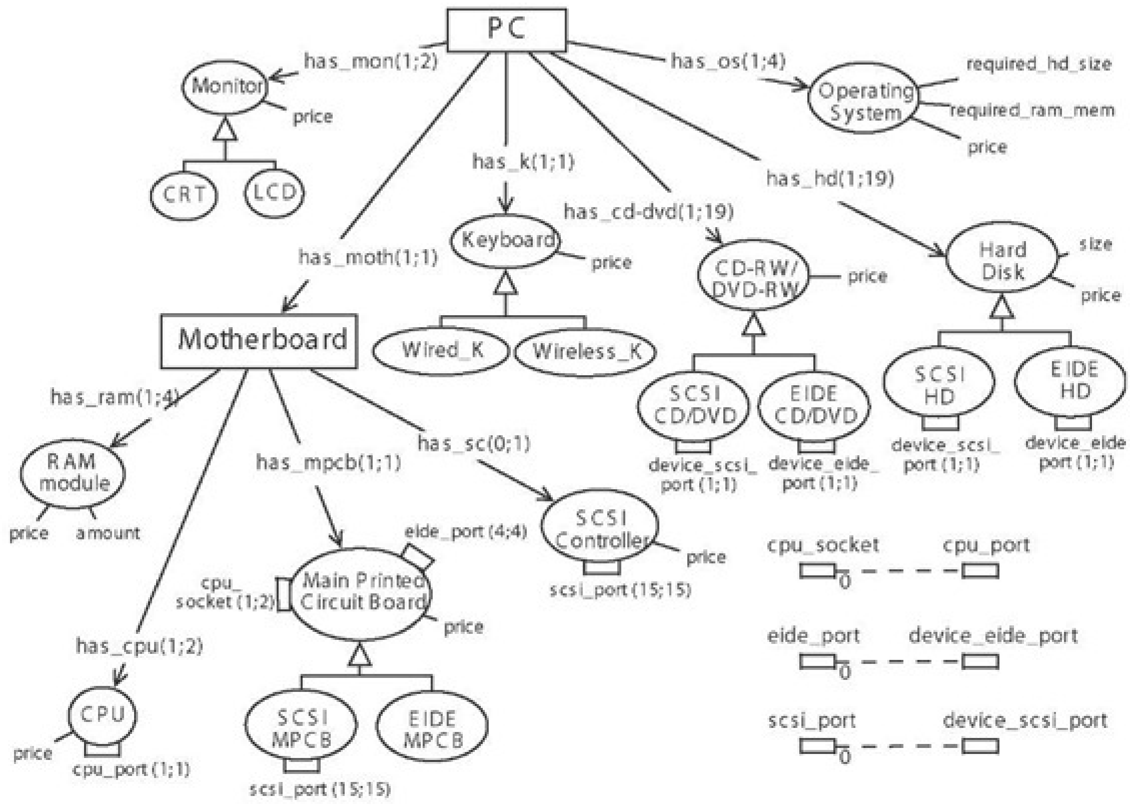
• **attributes used to** describe components, ports, resources and functions;

• **taxonomies in** which component, port, resource and function types may be organized in;

• **constraints to** specify conditions that configurations must satisfy.

Figure 1 depicts a simplified fragment of the domain knowledge for PC configuration. It describes all the PC variants valid for the domain. Has-part relations

Figure 1. A fragment of a PC configuration knowledge base



Constraints

- In any PC, if there is any S CSI device, then there must be either a SCSI Main Printed Circuit Board or a SCSI Controller

- In any PC, there must be no more than four EIDE devices and no more than fifteen SCSI devices

- In any PC, the total hard disk space required by all the Operating Systems must be less than the size of hard disks

- In any PC, the R AM required by each Operating System must be less than the available R AM amount

- In any Motherboard, there cannot be both a SCSI Main P rinted Circuit Board and a S CS I Controller

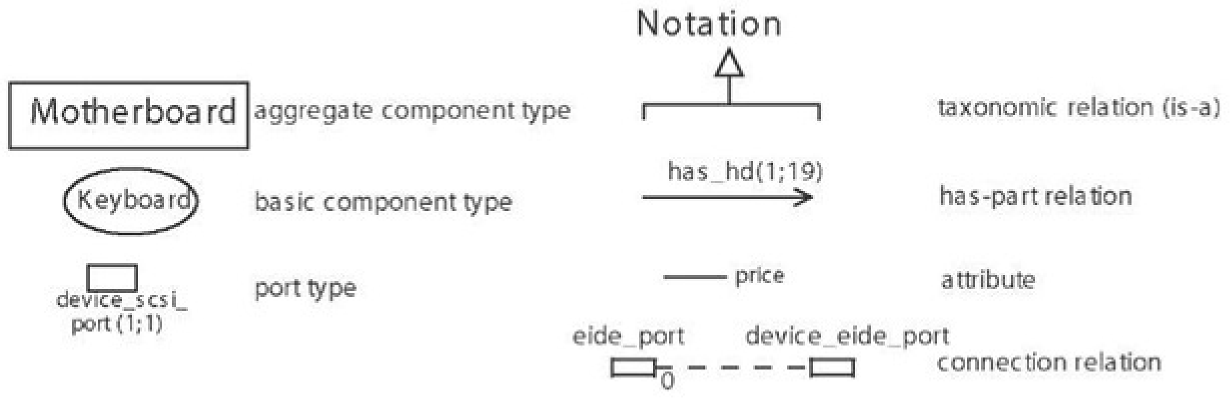


Figure 3 describes a particular PC variant, meeting the requirements stated in Figure 2 (containing also an optimization criterion on price).

Figure 2. An example of user requirements for a PC

R equirements

The PC should have:

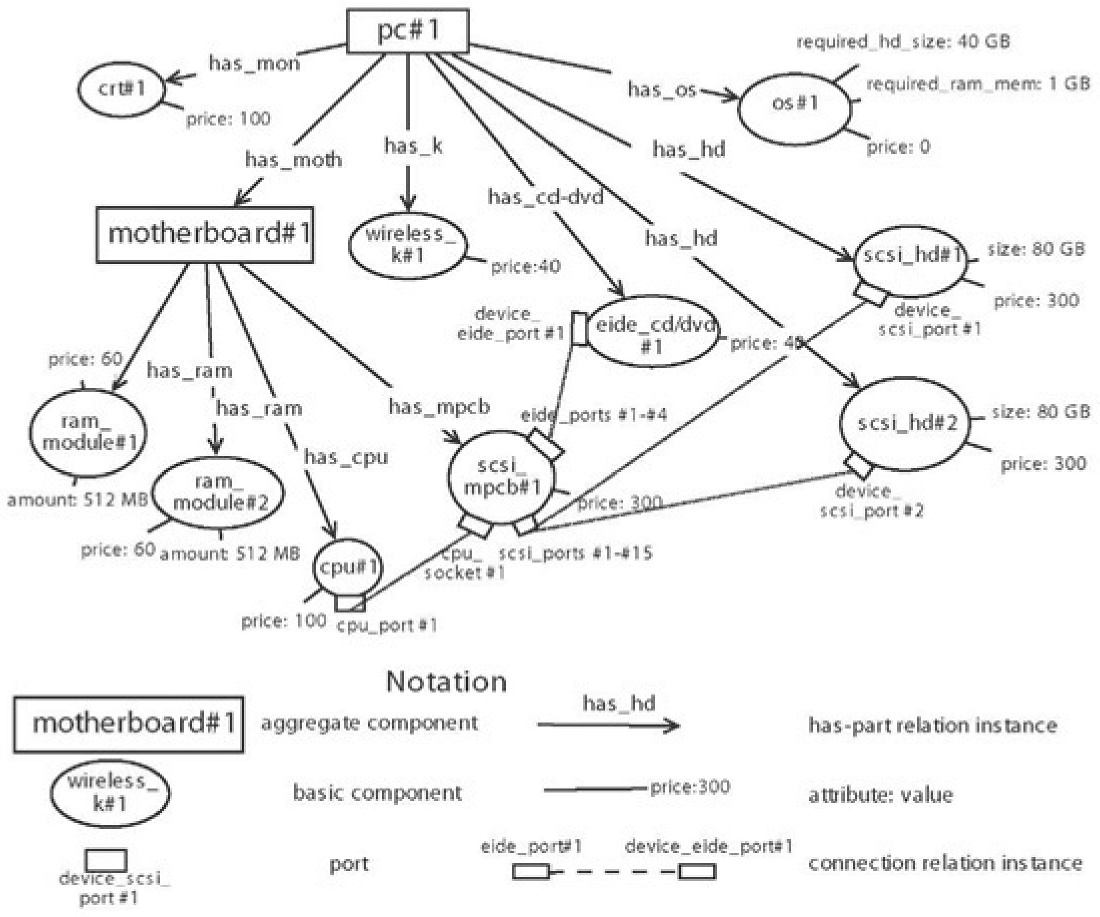
- two SCSI Hard Disks and atleast160 GB of Hard Disk

- atleastl GB RAM

- a Wireless Keyboard

- the cheapest price

Figure 3. A configured PC, compliant with the domain knowledge in Figure 1 and meeting the requirements in Figure 2



AUTOMATIC CONFIGURATION

Despite the consensus over the conceptualization of the problem, there is a wide range of approaches to configuration, with reference to different paradigms of Artificial Intelligence. It is possible to identify two mainstreams in current approaches to configuration: namely, constraint-based frameworks and logic-based frameworks. Constraint-based frameworks emphasize combinatorial aspects of configuration problems which have large search spaces and few solutions, while logic-based frameworks, in general, stress the description of the compositional structure of the product.

As regards constraint-based frameworks, approaches based on Constraint Satisfaction Problem (CSP) (Dechter, 2003) and its extensions are widely adopted. In particular, the classical CSP paradigm has been extended to overcome some of its limitations. In fact, on the one hand, in classical CSP, the set of variables is fixed and they will all be assigned values in every solution. On the other hand, in the configuration task the number and the types of the components that will be part of the final valid configuration are usually not known in advance, since they are selected by the configurator during the configuration process.

This fact motivated the introduction of Dynamic CSP (DCSP) (Mittal & Falkenhainer, 1990) (Gelle & Sabin, 2006) (also known as Conditional CSP) paradigm. A DCSP is defined – as classical CSP – on a fixed set of variables, but – differently from classical CSP – during the problem solving phase, it only takes into account the subset of variables relevant to the solution (i.e., the active variables). DCSP formalizes the notion of a particular type of constraint, i.e., activity constraints, which can add or remove variables from a potential solution depending on conditions imposed on already active variables. The search process starts with an initial set of active variables, and additional variables are introduced (or explicitly left out) as search progresses, depending on satisfied activity constraints.

A generalization of the original DCSP proposal, as well as some results on complexity and expressiveness, are presented in (Soininen, Gelle & Niemela, 1999). Several solving methods for DCSP are described and discussed in (Gelle & Sabin, 2003).

In (Stumptner & Haselbock, 1993) the authors further extend DCSP by introducing the Generative CSP. In Generative CSP the types of the components may be compactly described and managed; moreover, generic constraints are defined: these are constraint schemata which can be instantiated on the specific variables activated at a particular point in the configuration process.

In (Sabin & Freuder, 1996) the authors overcome a second major limitation of CSP with regard to configuration problems: in fact, CSP is “flat”, i.e., it does not allow to represent the structure of a configuration product in a straightforward way. To overcome this limitation, Sabin and Freuder propose Composite CSP, an extension to CSP which allows one to take into account not only changing sets of components, but also the hierarchical structure of the final configurations. In Composite CSP variables take not only atomic values but also values representing entire subproblems. Whenever a variable is assigned with a subproblem value, the subproblem is “expanded” and the problem is dynamically modified: specifically, it is “refined” by considering also the variables and the constraints in the subproblem. In such a way, it is easy to adapt the CSP’s inferential mechanisms to Composite CSP.

Also classical CSP itself plays an important role in configuration. In fact, in (Aldanondo, Moynard & Hamou, 2000) an approach is presented that uses standard CSP techniques to solve configuration problems. Moreover, several results in the configuration research field somehow refer to standard CSP framework. For example, in (Freuder, Likitvivatanavong & Wallace, 2001) the authors explore the problem of generating explanations for configuration problems expressed as CSP. In (Freuder & O’Sullivan, 2001) the authors propose an approach for dealing with configuration problems expressed as CSP where it is not possible to satisfy all user requirements at the same time, and it is necessary to establish a satisfactory trade-off between them. (Amilhastre, Fargier & Marquis, 2002) extends CSP to offer support for interactive problem solving as in the case of interactive product configuration, where the interactivity refers to the user making choices during the configuration process. Specifically, the approach provides the user with features such as consistency maintenance (i.e., inconsistencies are discovered as soon as possible), consistency restoration (i.e., guidance for relaxing inconsistent choices) and explanations (i.e., minimal sets of inconsistent choices are identified). Finally, (Freuder, Carchrae & Beck, 2003) describes an approach for removing values of variables in a CSP that would lead to a dead-end in solving the CSP.

As regards logic-basedframeworks, (McGuinness, 2002) analyzes Description Logics (DL) (Baader, Calvanese, McGuinness, Nardi & Patel-Schneider, 2003) as a convenient modeling tool for configurable products. DL make possible a description ofthe configuration knowledge by means of expressive conceptual languages with a rigorous semantics, thus enhancing knowledge comprehensibility and facilitating knowledge re-use. Furthermore, the powerful inference mechanisms currently available can be exploited both off-line by the knowledge engineers and on-line by the configuration system. Moreover, the paper describes a commercial DL-based family of configurators developed by AT&T.

(Soininen, Niemela, Tiihonen & Sulonen, 2000) describes an approach in which the domain knowledge is represented with a high-level language and then mapped to a set of weight constraint rules, a form of logic programs offering support for expressing choices and both cardinality and resource constraints. Configurations are computed by finding stable Herbrand models of such a logic program.

(Sinz, Kaiser & Kuchlin, 2003) presents an approach particularly geared to industrial context (in fact, it has been developed to be used by DaimlerChrysler for the configuration of their Mercedes lines). In Sinz et al.'s approach the domain knowledge is expressed as formulae in propositional logic; then, it is validated by running a satisfiability checker, which can also provide explanations in case of failure. However, this work aims at validating the knowledge base, rather than solving configuration problems.

There are also hybrid approaches that reconcile constraint-based frameworks and logic-based frameworks. For example, both (Magro & Torasso, 2003) and (Junker & Mailharro, 2003) describe hybrid frameworks based on a logic-based description of the structure of the configurable product (taking inspiration from logical languages derived from frame-based languages such as the DL) and on a constraint-based description of the possible ways of interaction between components.

In (Junker & Mailharro, 2003) constructs of DL are translated into concepts of constraint programming in order to solve a configuration problem. On the contrary, (Magro & Torasso, 2003) adopts an inference mechanism specific for configuration, which, basically, searches for tree-structured models on finite domains for conceptual descriptions, and adapts some constraint-propagation techniques to the logical framework.

In most formalizations, the configuration task is theoretically intractable (at least NP-hard, in the worst case) and in some cases the intractability does appear also in practice and solving configuration problems can require a huge amount of CPU time. There are several ways that can be explored to cope with these situations: providing the configurator with a set of domain-specific heuristics, defining general focusing mechanisms (Magro & Torasso, 2001), making use of compilation techniques (Sinz, 2002) (Narodytska & Walsh, 2006), re-using past solutions (Geneste & Ruet, 2002), defining techniques to decompose a problem into a set of simpler subproblems (Magro, Torasso & Anselma, 2002) (Anselma & Magro, 2003).

Configuration has a growing commercial market. In fact, several configurator systems have been developed and some commercial tools are currently available (e.g., ILOG (Junker & Mailharro, 2003), Koalog, OfferIt! (Bergenti, 2004), Oracle, SAP (Haag, 2005), TACTON (Orsvarn, 2005)).

Furthermore, some Web sites have been equipped with configuration capabilities to support customers in selecting a suitable product in a wide range of domains such as cars (e.g., Porsche, Renault, Volvo), bikes (e.g., Pro-M Bike Configurator) and computers (e.g., Dell, Cisco).

FUTURE TRENDS

Many current configuration approaches and software configuration systems concern the configuration of mechanical or electronic devices/products and are conceived in order to be employed by domain experts, such as production or sales engineers.

Nowadays, the scope of configuration is growing and the application of automatic configuration techniques to non-physical entities is gaining more and more importance. The configuration of software products and complex services built on simpler ones are two research areas and application domains that are currently attracting the attention of researchers.

The capability of producing understandable explanations for their choices or for the inconsistencies that they encounter and of suggesting consistency restorations are some needs that configuration systems share with many knowledge-based or expert systems. However, the aim of making configuration systems profitably used by non-expert users too and the deployment of configurators on the Web all contribute to strengthen the importance of these issues.

Explanations and restorations are also related to the topic of interactive configuration, which is still posing some challenging problems to researchers. Indeed, besides these capabilities, interactive configuration requires also effective mechanisms to deal with incomplete and/or incremental requirements specification (and with their retraction) and it is also demanding in terms of efficiency of the algorithms.

Real-world configuration knowledge bases can be very large and they usually are continually modified during their life cycle. Some research efforts are currently devoted to define powerful techniques and to design and implement tools that support knowledge acquisition, knowledge-base verification and maintenance.

Furthermore, a closer integration of configuration into the business models and of configurators into enterprise software systems is an important goal for several companies (as well as for enterprise software providers).

Distributed configuration is another important topic, especially in an environment where a specific complex product/service is provided by different suppliers that have to cooperate in order to produce it.

Finally, it is worth mentioning the reconfiguration of existing systems, which is still mainly an open problem.

CONCLUSION

Configuration has been a prominent area of Artificial Intelligence since the early Eighties, when it started to arouse interest among researchers working in academia and industry.

This article provides a general overview of the area of configuration by introducing the problem of configuration, briefly presenting a general conceptualization of configuration tasks, and succinctly describing some representative proposals in literature to deal with configuration problems.

As we have illustrated, during the last few years several approaches involving configuration techniques have been successfully applied in order to deal with issues pertaining to a wide range of real-world application domains, ranging from cars to computer systems, from software to travel plans.

Theoretical results achieved by the academic environment have found effective, tangible applications in industrial settings, thus contributing to the diffusion of both industrial and commercial configurators. Such applications – in their turn – gave rise to new challenges, engendering a significant cross-fertilization of ideas among researchers in academia and in industry.

KEY TERMS

Constraint Satisfaction Problem (CSP): A CSP is defined by a finite set ofvariables, where each variable is associated with a domain, and a set of constraints over a subset of variables, restricting the possible combinations of values that the variables in the subset may assume. A solution of a CSP is an assignment of a value to each variable that is consistent with the constraints.

Description Logics (DL): Logics that are designed to describe concepts and individuals in knowledge bases. They were initially developed to provide a precise semantics for the frame systems and the semantic net- works. The typical inference for concepts is checking if a concept is more general than (i.e., subsumes) another one. The typical inference for individuals is checking if an individual is an instance of a concept. Many DL are fragments of first-order logic, while some of them go beyond first order.

Logic Program: A logic theory (possibly containing some extra-logic operators) that can be given a procedural meaning such that the process of checking if a formula is derivable in the theory can be viewed as a program execution.

Mass Customization: A business strategy that combines the mass production paradigm with product personalization. It is closely related to the modularity in product design. This design strategy makes it possible to adopt the mass production model for standard modules, facilitates the management of product families and variants and it leaves room for (various kinds and degrees of) personalization.

Production-Rule-Based System: A system where knowledge is represented by means of production rules. A production rule is a statement composed of conditions and actions. If data in working memory satisfy the conditions, the related actions can be executed, resulting in an update of the working memory.

Propositional Logic Formula Satisfiability: The task of checking whether it is possible to assign a truth value to every variable that occurs in a propositional formula, such that the truth value of the whole formula equals true.

Stable Herbrand Model: A minimal set of facts satisfying a logic program (theory). Each fact in the model is a variable-free atom whose arguments are terms exclusively built through function and constant symbols occurring in the program and whose predicate symbols occur in the program as well. Facts not appearing in the model are regarded as false.

Next post: Constraint Processing (Artificial Intelligence)

Previous post: Conditional Hazard Estimating Neural Networks (Artificial Intelligence)

- **Related Links**
  - Artificial Intelligence
    - Active Learning with SVM (Artificial Intelligence)
    - Adaptive Algorithms for Intelligent Geometric Computing (Artificial Intelligence)
    - Adaptive Business Intelligence (Artificial Intelligence)
    - Adaptive Neural Algorithms for PCA and ICA (Artificial Intelligence)
    - Adaptive Neuro-Fuzzy Systems (Artificial Intelligence)
- **:: Search WWH ::**

Search

Google Custom Search