# Solving Configuration Problems in Excel :

## How to successively address Professional Markets through the Office world

## MSR-TR-2006-106

Youssef Hamadi and Lucas Bordeaux
Microsoft Research Ltd.,
7 J J Thomson Avenue Cambridge CB3 0FB,
United Kingdom,
{youssefh, lucasb}@microsoft.com

24th July 2006

**Abstract**

This work shows how Excel can be leveraged to tackle configuration problems. We demonstrate this by presenting the full development cycle of a PC-configurator prototype. We articulate our demonstration by exemplifying several important points. First, that Excel can be used to import raw products and services data. Second, that these data can be used by a Constraints configuration modelling to correctly instantiate a set of configuration rules. Third, that any update to the data like addition of new components can be performed without altering the modelling. Fourth, that dedicated Excel GUIs can be used for interactive product and service configuration. Finally, we claim that the proposed extension would become a nimble way to successfully address professional markets (B2B, B2C). Indeed, the pervasive nature of the Office suite would make any Office-based Business Solution widely adopted by a large basis of Office customers.

**Keywords: Excel, Office, Configuration problems, B2B/B2C**

# 1 Introduction

Product configurators are helping salespersons and partners to successfully match customers' requirements to unique products and service offerings. These tools are critical since they greatly reduce the complexity and therefore improve productivity. We can distinguish between several configuration problem classes : ship-to-order (STO), assemble-to-order (ATO), engineer-to-order (ETO), and Internet Pricing and Configuration (IPC).

STO products have little variability other than a predetermined set of attributes, such as color and size. Some office equipment, household appliances and televisions

fall into this classification. ATO products are configurable offerings made up of standard components. They are configured based on customer needs and intercomponent attribute relationships (such as compatibility). Computers and telecommunications equipment fall into this classification. ETO products have the same features as ATO products, but involve some level of engineering analysis to be configured. Finally, non-product based configuration like IPC enables selling channels to deploy customized trade promotions and implement complex pricing and discounting strategies [Des06].

In order to be successful, product/service configuration must rely on a model that enables users to efficiently enter their needs, integrates with other entreprise systems, and generates outputs such as a line item in a quotation or a bill of material for manufacturing processes. Technically, this model is used by an automated tool like a Constraint Solver whose algorithms are used to perform proactive consistency check of the modelling with respect to the select and deselect actions performed by end-users. Moreover these algorithms can also automatically "finish" an order by completing remaining choice at any time during the interaction. When cost or quality data are available, optimization algorithms can automatically perform the choices which minimize some clearly defined cost function e.g., components with cheapest overall cost in ATO, percentage choice which minimizes duration of financing plans in IPC, etc.

In this paper, we claim that Microsoft Excel has all the required flexibility to implement efficient configurators. To support this claim we are going to demonstrate various points. The most critical being the programmatic integration of a Constraint Solver within Excel. Here the goal is to demonstrate that constraint modelling can effectively be performed without violating Excel's programmatic model. We also need to show that raw business data which characterize products and services can easily be integrated and addressed by Excel's Constraints modelling. Finally, we must show that classical interactive configuration is possible through Excel GUIs.

Moreover, we argue that adding such Configuration capability to an Office product would have a great impact on Microsoft strategy. Indeed, it would allow the company to fight big competitors (Oracle, SAP, Siebel, etc) with its greatest assets: a large basis of pre-installed Office systems.

The paper is articulated as follow: Section 2.1 overviews the functionalities we propose to add to Excel; the core of the paper is the presentation of an example of configuration problem modelled and solved using the enhanced software (Section 2); we then discuss the proposal and conclude in Section 3.

# 2 Solving Configuration Problems in Excel

## 2.1 Adding a Constraint Programming Engine

We briefly describe a number of functionalities which we propose to add to Excel. The basic idea is to use MSRC's Constraint Solver[1], Disolver to enforce configuration rules

[1]Constraint solvers should not be confused with linear programming (LP) solvers such as the one which is readily available in Excel; the tools share similarities but constraint programming deals with more general types of constraints, such as Boolean constraints, which cannot be expressed naturally in LP and are needed to model business rules (see Section 2.2.2).

[Ham03]. Presenting Disolver and the Constraint Programming technology in general goes beyond the scope of this paper; to understand the proposal, all the reader needs to know is that the following functionalities are provided by Disolver, and that their integration in Excel is essentially a matter of wrapping them as new Excel functions:

- Basic functions for the creation of *decision variables*. A decision variable represents an unknown quantity ranging over a finite number of values. In Excel, a decision variable will be attached to a cell; the add-on we need is simply a new function which specifies that the content of the cell is the value of a constraint programming decision variable with a given range. For instance, a cell containing the definition `=CP_Var("X", C1:C10)` will correspond to a variable, whose internal name is "X" and, whose value can be any of the values contained in the specified range of Excel's cells.

- Basic functions for the creation of *constraints* on these variables. A constraint is a relation imposed on some of the decision variables. It is contained by a cell and relates to a number of variables or constraints cells. Its truth value (*i.e.*, whether it is satisied or not) gives the value of its cell and can be reused by other functions. The difference with a classical Excel function however, is that the constraint can be *imposed*, *i.e.* we can ask the solver to ensure that it always evaluates to true. Naturally, Disolver's algorithms will then need to find satisfactory values for the variables related by the constraint. In a configuration application, the constraints will be used to specify the *business rules* of the application.

As we can see, we can add the basic building blocks of Constraint Programming without violating Excel's programmatic model. Indeed, decision variables and constraints operate in cells and use cells as input. Moreover, they return values and can therefore be mixed consistently with classical Excel's functions and algorithms. These functionalities are illustrated in more details in the next part.

## 2.2 Development Cycle of a Typical Application

Our goal in this section is to show that Excel, extended with the previous functionalities, would become a powerful tool for modelling and solving configuration problems. For the ease of presentation, we consider a classical "PC Configuration" benchmark widely used in the research community [Cli], and which falls under the ATO class. As its name suggests, this problem concerns the configuration of PCs which can be built using several types of processors, mother boards, graphic cards and other components. Some components are incompatible, for instance a mother board of type "Abit BX6 ATX" is compatible with a "Slot1" CPU slot but not with other types of CPUs, *etc*. The goal is to find a configuration which is compatible with the desiderata of the user. The application helps in two transparent ways: (1) when the user imposes a choice for a particular component, or forbids some values for this component, the choices for other components which become incompatible are automatically discarded; this is the *propagation* mode; (2) when the user has specified all her desiderata, she can click a "solve" button and the application will automatically propose her with a fully-specified

solution satisfying her requirements (for instance, if the user did not specify any preference regarding the bus component, the application will fill-in the corresponding cell with a value consistent with the other choices); this is the *resolution* mode.

### 2.2.1 Data Integration

An efficient configurator must provide an easy way to integrate new data such as new product components which should be entered without altering the configuration modelling and the presentation layer parts.

Excel can easily integrate raw data and present them in a column-based form. This is exemplified in figure 1 which presents raw data for some PC-configurator application. Each column is tagged with the name of the component and populated with the available parts.

### 2.2.2 Configuration Modelling

In Constraint Programming, modelling involves the definition of constraints restricting the possible combinations of the variables of a problem. Therefore we need to show that we can easily express CP variables in Excel and easily restrict their range through constraints.

**Decision variables** A discrete decision variable represents a set of alternative choices which will be eventually pruned through constraint propagation and/or end-users decisions. For instance a variable can represent the *mother-board* component and its domain can be made of potential choices, {*AopenAK-72133ATX*, *AsusK7MATX*, *Microstar6167ATX*, *etc.*}. This range directly addresses the data integration worksheet of figure 1. For example,

```
= CP_Var("MotherboardI",'PC components'!H:H)
```

defines a variable called “MotherBoard” (this is just an internal name useful for data output) and populates its domain with components data located in *H:H*. Here we can remark that the integration of new mother board components can be made without altering the modelling since the addition of new data in *H:H* will be seamlessly reported in the related variable domain.

This variable and all the decision variables of the PC configurator benchmark are defined in the *Variables* section of the worksheet presented in figure 2.

Since a variable is represented as a function it returns its value. This value essentially encodes the result assigned to the variable. Depending on which mode is being used (*i.e.* propagation or resolution), variables may be all instantiated or some of them may have a range of possible values. In this case, several choices are obviously possible: we can either display the range of allowed values (*e.g.* {*AopenAK-72133ATX*, *AsusK7MATX*}), or display a single proposal of value (emphasized with a special colour, so that the user sees that other values are possible for this variable), or simply display a special flag indicating that the variable is not instantiated yet.

**Business rules** We know how to define the decision variables which will encode valid configurations. We now need to define constraints which will represent the business rules expressing the correct combinations of decision variables.

A business rule is defined as a new Excel function which takes as input a collection of decision variables and maintains a specific invariant on them e.g., all the variables must have different values. As we have presented, CP variables are located in Excel cells and can therefore easily become the input of rules expressed as new Excel functions which will add constraints to the Disolver model. This is presented in the Constraints part of figure 2.

The first rule is related to the mother board components. It defines the list of accepted configurations through a list of conjunctions (presented in line 32). Each conjunction accesses decision variables and specifies a correct mother board realisation. For instance, we find in cell B32,

```
= (B18 = "Abit BX6 ATX" && B10 = Slot1 && B13 = IDE
&& B23 = 4 && B15 = "233MHz"  && B16 = "450MHz"
&& B12 = SDRAM168Pin  && B17 = 4 && B26 = PCI
&& B27 = PCI && B28 = PCI  && B29 = AGP)
```

This rule validates a configuration using an "Abit BX6 ATX" mother board combined to a "Slot1" CPU slot, etc. The 13 possible mother board configurations of the PC benchmark can be defined with as many conjunction constraints ultimately combined in a disjunctive formula. This is presented in B33,

```
= Impose(B32 || C32 || ...)
```

This constraint imposes a disjunction between rules B32, C32, etc. It takes as input the previous set of rules. As said before, a constraint returns its truth value[2] which can be reused by some other rule, or imposed in a constraint. Valid Processor components are similarly defined in the bottom of figure 2 through a simple disjunction of conjunctions.

Remark that we do not give all the rules of this model which also contains conjunction of disjunction which specify correct combinations of all the remaining parts of a proper PC[3].

### 2.2.3 Interactive Application

Excel GUIs can easily use our configuration modelling to provide an interactive front-end to end-users. An example of such a front-end for our PC configuration problem is presented in figure 3. It is made of several combo-boxes which report the domains of the decision variables. Each select or deselect action is reported to the modelling

[2]Note also that this value is not necessarily fully specified if we are in propagation mode. In this case, one option is to return the set $\{0, 1\}$, reflecting the fact that both values 0 (false) and 1 (true) are possible for the rule.

[3]The full Disolver modelling for this benchmark and other classical configuration benchmark is available upon request.

where business rules are enforced through constraints and eventually lead to value suppressions or enforcement in other decision variables.

Combo-boxes can be addressed in any order with the insurance[4] of moving in a consistent space of configurations. When a choice leading to an inconsistency is performed, it can be easily reported through some dialog box with some clear explanation of the conflict.

At any time, a *Solve* button can be pressed to finish a partial configuration. In optimization settings we can imagine several Solve button related to different optimization criterion, e.g., Solve/cheapest, etc. Additionally, an interaction can be saved for later access through appropriate buttons.

While a final user, salesperson or customer would have a frequent access to the interface of the application, the modelling part (e.g., figure 2), would typically be written by an Excel programmer and would not change frequently. Eventually, new components type could be added or deleted and this will require a refresh of the data worksheet (e.g., figure 1). Importantly, this will not require a change in the modelling part.

Overall, solving configuration problems in Excel is possible, easy and worthwhile. It is possible thanks to the integration of a Constraints solver and it is easy because configuration modelling can be performed in Excel's programmatic model : without changing the habits of an average Excel programmer. Finally, it is worthwhile since the programming part can consistently cope with frequently changing business data.

# 3 Conclusion and Discussion

In this paper, we showed how Excel can be leveraged for configuration problems. Beside the simple addition of a new capability to a flagship product, we think that the proposed extension would be a nimble way to address B2B and B2C markets. Indeed, Office is pervasive and any Office-based Business Solution would become widely adopted by a large basis of Office customers. Furthermore, because Excel is so popular, we believe that our proposal could indeed contribute to a wider adoption of configuration technologies, since it is conceivable that more users would get familiar with it once it is available in the Office package, and would then get interested in more specialised offers as their use of configuration tools increases.

We are currently implementing the proposed integration which will be available upon request. However, important questions remain.

A first question is whether the added Constraint Programming functionalities can serve other purposes than configuration applications. A central feature of CP is that it expresses *relational* dependencies between data. Excel, on the other hand, is based on the more traditional notion of *functional* dependency. For instance we can specify that cell *Benefits!C5* is equal to the sum of all numbers of column $F$. Therefore, the propagation is just one-way: modifying the value of one cell in column $F$ will propagate to cell *C5*. Adding relational dependencies allows to propagate the dependencies backward and to automatically deduce, for instance, values or intervals of values for

[4]Up to current consistency level.

the cells of column $F$ which are not filled yet, given a prospective value for $C5$. We think that this functionality may very well be of interest *per se* since it would allow the user to interactively simulate how modifications in some of her data impact other data.

A second question is related to the fact that ETO can involve some level of engineering analysis to configure products. It would be interesting to see if some of these engineering tasks could be performed from an Office product closely linked with the configurator e.g., Excel, Visio, etc. If the answer is positive that would become another strong point for our proposal.

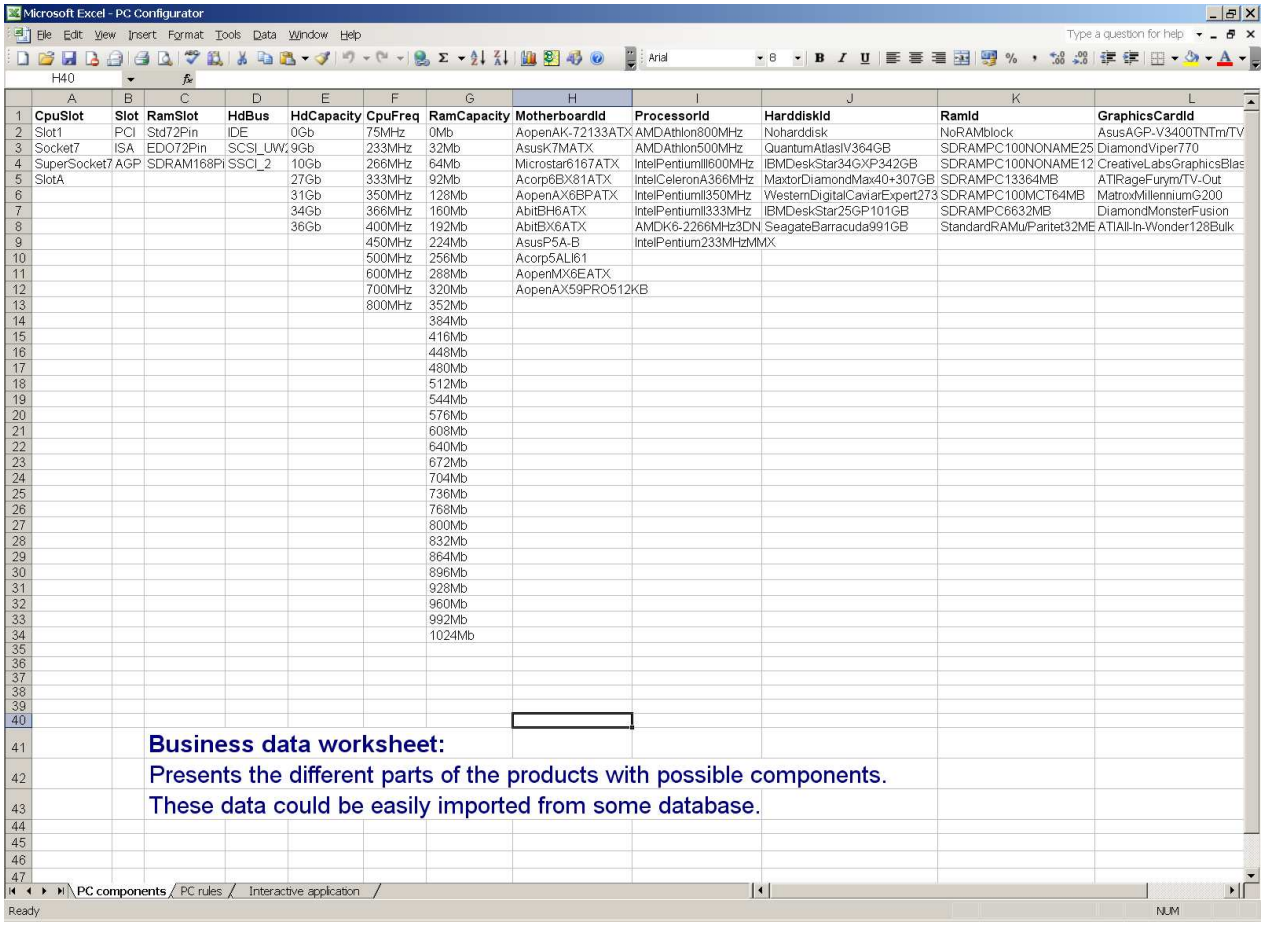
Altogether, it seems that Excel-based configurators would be perfect to address small and medium problems currently solved by small companies operating in niche markets. Again, the main advantages for customers would be cheap cost and system integration. Large markets with fierce competition could still rely on dedicated Microsoft products doing configuration in complex B2B/B2C settings. However, any Microsoft offer should be unified and here, the use of the same Constraints solving technology which involves the acceptance of the same problem solving formalism would greatly help. Indeed, it would accompany and ease customers' evolution from small Excel based configurators defined with extended spreadsheet constructs to Microsoft standalone applications coded in some high level programming language.

# References


[Cli] Clib:configuration-benchmarks-library. http://www.itu.dk/research/cla/externals/clib/.

[Des06] R. Desisto. Marketscope for sales configuration, 2Q06. Technical Report n/a, Gartner, 2006.

[Ham03] Y. Hamadi. Disolver : A Distributed Constraint Solver. Technical Report MSR-TR-2003-91, Microsoft Research, Dec 2003.




| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | **CpuSlot** | **Slot** | **RamSlot** | **HdBus** | **HdCapacity** | **CpuFreq** | **RamCapacity** | **MotherboardId** | **ProcessorId** | **HarddiskId** | **RamId** | **GraphicsCardId** |
| 2 | Slot1 | PCI | Std72Pin | IDE | 0Gb | 75MHz | 0Mb | AopenAK-72133ATX | AMDAthlon800MHz | Noharddisk | NoRAMblock | AsusAGP-V3400TNTm/TV |
| 3 | Socket7 | ISA | EDO72Pin | SCSI_UW | 9Gb | 233MHz | 32Mb | AsusK7MATX | AMDAthlon500MHz | QuantumAtlasIV364GB | SDRAMPC100NONAME25 | DiamondViper770 |
| 4 | SuperSocket7 | AGP | SDRAM168Pi | SSCI_2 | 10Gb | 266MHz | 64Mb | Microstar6167ATX | IntelPentiumIII600MHz | IBMDeskStar34GXP342GB | SDRAMPC100NONAME12 | CreativeLabsGraphicsBlas |
| 5 | SlotA | | | | 27Gb | 333MHz | 92Mb | Acorp6BX81ATX | IntelCeleronA366MHz | MaxtorDiamondMax40+307GB | SDRAMPC13364MB | ATIRageFurym/TV-Out |
| 6 | | | | | 31Gb | 350MHz | 128Mb | AopenAX6BPATX | IntelPentiumII350MHz | WesternDigitalCaviarExpert273 | SDRAMPC100MCT64MB | MatroxMillenniumG200 |
| 7 | | | | | 34Gb | 366MHz | 160Mb | AbitBH6ATX | IntelPentiumII333MHz | IBMDeskStar25GP101GB | SDRAMPC6632MB | DiamondMonsterFusion |
| 8 | | | | | 36Gb | 400MHz | 192Mb | AbitBX6ATX | AMDK6-2266MHz3DN | SeagateBarracuda991GB | StandardRAMu/Paritet32ME | ATIAll-In-Wonder128Bulk |
| 9 | | | | | | 450MHz | 224Mb | AsusP5A-B | IntelPentium233MHzMMX | | | |
| 10 | | | | | | 500MHz | 256Mb | Acorp5ALI61 | | | | |
| 11 | | | | | | 600MHz | 288Mb | AopenMX6EATX | | | | |
| 12 | | | | | | 700MHz | 320Mb | AopenAX59PRO512KB | | | | |
| 13 | | | | | | 800MHz | 352Mb | | | | | |
| 14 | | | | | | | 384Mb | | | | | |
| 15 | | | | | | | 416Mb | | | | | |
| 16 | | | | | | | 448Mb | | | | | |
| 17 | | | | | | | 480Mb | | | | | |
| 18 | | | | | | | 512Mb | | | | | |
| 19 | | | | | | | 544Mb | | | | | |
| 20 | | | | | | | 576Mb | | | | | |
| 21 | | | | | | | 608Mb | | | | | |
| 22 | | | | | | | 640Mb | | | | | |
| 23 | | | | | | | 672Mb | | | | | |
| 24 | | | | | | | 704Mb | | | | | |
| 25 | | | | | | | 736Mb | | | | | |
| 26 | | | | | | | 768Mb | | | | | |
| 27 | | | | | | | 800Mb | | | | | |
| 28 | | | | | | | 832Mb | | | | | |
| 29 | | | | | | | 864Mb | | | | | |
| 30 | | | | | | | 896Mb | | | | | |
| 31 | | | | | | | 928Mb | | | | | |
| 32 | | | | | | | 960Mb | | | | | |
| 33 | | | | | | | 992Mb | | | | | |
| 34 | | | | | | | 1024Mb | | | | | |
| 35 | | | | | | | | | | | | |
| 36 | | | | | | | | | | | | |
| 37 | | | | | | | | | | | | |
| 38 | | | | | | | | | | | | |
| 39 | | | | | | | | | | | | |
| 40 | | | | | | | | | | | | |
| 41 | | | | | | | | | | | | |
| 42 | | | | | | | | | | | | |
| 43 | | | | | | | | | | | | |
| 44 | | | | | | | | | | | | |
| 45 | | | | | | | | | | | | |
| 46 | | | | | | | | | | | | |
| 47 | | | | | | | | | | | | |

Figure 1: Integration of components data

Figure 2: Definition of the configuration modelling through access to the Constraints solver components

Figure 3: End-users interactive application