Decision Oriented Programming in HCI: the Multi-Attribute Decision Language MADL

Bjoern Zenker

Chair for Artificial Intelligence
Friedrich-Alexander-University Erlangen-Nuremberg
Haberstraße 2, D-91058 Erlangen
bjoern.zenker@cs.fau.de

Abstract. In Human Computer Interaction (HCI), the computer has to take many decisions to react in a way that human wants. As decisions in HCI are diverse, contradictory, and hard to measure it is hard to study and model them, e.g. finding a mapping from the users preferences to adaptions of the user interface. To ease these tasks, we developed MADL (Multi-Attribute Decision Language). This programming language, based on Multi-Attribute Decision Making (MADM), is designed to model and make hierarchical multi-attributive decisions and is based on the analysis of decisions and goals in HCI. It fosters respecting HCI specific characteristics in development, like uncertainty and risk, can be embedded easily in other applications and allows inclusion of and experimentation with different decision rules. User interface logic can also be modeled by non-programmers and more easily separated from the business logic. The applicability will been shown in three use cases.

Key words: Multi-Attribute Decision Making, Human Computer Interaction

1 INTRODUCTION

Computers should react the way as humans expect them to. But still in many cases, computers do not react the way that humans expect them to. Thus there is often a discrepancy between the reaction of the computer and the action anticipated by the user. We expect this to be caused in a lacking knowledge of the computer about the users various goals and not taking these goals into account.

As machines and computers are going to be used by more and more users, and become more integrated in our daily lives, methods are needed to further tail the machines behavior to the users preferences. To further study and and undertake a step to close this gap we designed the programming language MADL to easily model tasks in Human Computer Interaction (HCI) and make multi-attribute decisions (for an introduction see e.g. [1]) on them. Our goal was the creation of an environment for decision taking, which hides complexity and different algorithms behind an intuitive programming language, to allow inclusion of decisions modeled by non-programmers into programs. Furthermore, we wanted to create a platform for experimentation and teaching in multi-attribute decision making.

As far as we know, no general tools for describing and making multi-attribute decisions with risk and uncertainty exist. But two similar systems from the HCI point of view, which also cover modeling of tasks, should be mentioned.

GOMS (Goals, Operators, Methods and Selection) [2], a technique for measuring and analyzing tasks in HCI, is very similar in structure to MADL. GOMS divides the user's interaction into goals, sequences (methods) of actions (operators) and selection rules, which model which sequence would be selected by a user. As we will see, these rules resemble decisions in MADL. But as the focus of GOMS is on analysis of HCI, selection rules are often ignored in GOMS. According to the *Kleindorfer* classification of decision theories [3] GOMS is *descriptive*. In contrast, MADL is *prescriptive*, with the main focus – in GOMS terms – on selection, to shape HCI, rather than to analyze it.

CTT (Concurrent Task Tree) [4] is a formalized task model which models tasks in HCI, with the addition to also allow representation of concurrency besides tasks. CTT is also descriptive, with a focus on modeling tasks.

In contrast to the presented systems, the focus of MADL is on solving decision problems.

After a short overview of decisions in HCI and their relation to human preferences (section 2) this paper introduces some aspects of the MADL language design (section 3) and it's runtime environment. Results of a proof-of-concept study for MADL usage by programming a virtual assistive video-recorder, will be shown (section 4). Finally we summarize our results and give an outlook on future work (section 5).

2 GOALS AND DECISIONS IN HCI

2.1 Goals

Systems with HCI have despite their variety something in common: the interaction between human and machine serves in all cases the best possible satisfaction of preset goals. According to [5] goals are various, exhibit a hierarchical structure and have complex relations between each other, which means that goals may also be contradictory! Principles and definitions of usability, which is closely related to goals in HCI, are presented by [6], [7] and [8]. Studying the lists of goals from the other authors leads us to the following summarization: The goals are mostly abstract. No methods exist for measuring the goals. There is a deep structure of goals. Relations between goals are unclear and possibly contradictory. This leads us to the conclusion, that there is currently no formal definition of goals and their relations in HCI. To further examine this lack of formality, or rather to bridge the gap between the unformal field of human goals and the formal field of computation, we developed MADL.

2.2 Decisions

With respect to the above mentioned goals, decisions have to be taken in HCI.

Examinations of decisions in HCI lead us to three main kinds of decisions in HCI.

Decisions for interaction control the way, how the machine interacts with the user. [9] gives a list of adaptions for the user interface, like content adaption, selection of information, quantity of information, augmentation of information, layout of information, modality of information and dialog adaption.

Decisions for automation control the technical process of the machine. The main question here is the grade of automation: which processes and tasks should be overtaken be the machine and which not. [7] gives an overview, in which areas are machines are superior to humans and vice versa.

Metadecisions control the act of decision making per se. They include decisions like, what to do in stand-off situations, which risks should be taken and whether more information should be retrieved in order to improve the decision quality. Criteria of these decisions are mostly very special, but due to existing theories like information theory (e.g. [10]), risk theory (e.g. [11]) and Dempster–Shafer theory (e.g. [12]) easy to measure and calculate.

2.3 Erwartungsstrukturen

According to [11] there exist mainly three "Erwartungsstrukturen" (engl. structure of expectance) for decisions: decisions under security, where the future can be predicted; decisions under uncertainty in a closer sense, where no precise probability of the possible future scenarios exist; and decisions under risk, in which probability values can be assigned to the possible future scenarios. For usage in MADL, all three "Erwartungsstrukturen" can be reduced to one: decisions under security, can be seen as decisions under risk with all probability values 100%. Applying the principle of indifference, thus assuming a uniform distribution for the future scenarious, decisions under uncertainty in a closer sense can be expressed as decisions under risk. Thus, MADL will focus on decisions under risk. Furthermore we will only consider decisions with a finite set of alternatives, each alternative consisting of a finite ordered sequence of actions. Closed-loop control systems are not discussed in this paper.

As there are diverse decisions to take in HCI, the language to express them must be general.

3 MADL

3.1 System overview

A complete system to model and take decisions was built: The MADE (Multi-Attribute Decision Environment)[1] interpreter takes MADL programs as input,

[1] Download, online interpreter and MADL grammar available at http://www8.informatik.uni-erlangen.de/en/zenker.html

evaluates them and returns an ordered list of the best alternatives for a given decision. One of these alternatives can then be passed to a plan execution system. MADE may be used on the command line, as a graphical tool (ShowMADE) or as a Java package.

3.2 MADL program

A MADL program consists of several MADL-decisions and may be linked to *prognosticating models* (PM). The following code shows a basic MADL-program which will be explained in the subsequent sections.

```
CON examples.HelloWorldModel AS model
CON examples.RiskModel AS risk

ParetoDecision Casino {
  ALT [DO THIS, DO THAT]
  ALT [GO THERE, DO THIS]

  GOAL MAX!(model.fun)
  GOAL model.spendMoney < 1000
  GOAL MAX!(risk.averseToRisky(model.money, 10))
}
```

Prognosticating model A PM can be a arbitrary system, like Java methods, a Bayesian network or a Hidden Markov models, to simulate the "Erwartungsstruktur", that is to predict values of variables after the application of an alternative. We will call the set of predicted values for one alternative *context*. The two lines starting with `CON` import a model each.

MADL decision A MADL decision d is a quadruple $d(r, A, G, C)$, where r is the decision rule, A the set of alternatives, G the set of goals and C the set of constraints.

MADL decisions may be inherited from other MADL decisions. The child decision inherits decision rule, alternatives and goals, but more alternatives and goals may be added and the decision rule may be changed. The above example shows a decision named `Casino` with the `ParetoDecision` rule. The decision contains two alternatives, two goals and one constraint.

Actions A is a ordered sequence of actions $a \in A$. An action can be of several types:

normal action an action from the model with an arbitrary number of parameters, e.g. `DO THIS`
assignement action assignment of a new value to a context variable which will be propagated to the model, e.g. `model.fun = 100%`

best-of decision action import the n best alternatives from another decision, e.g. `!otherDecision n`

mean-of decision action import the mean value of consequences of the n best alternatives, e.g. `?otherDecision n`

expanding action replicates the alternative n times and appends one of the n best alternatives from another decision to each, e.g. `<otherDecision` (see section 3.3 for an example)

If the consequences of an action should only be considered in the decision it is specified in, it can be surrounded by parentheses. That way, the action will not be considered in inherited decisions. Each of the two above alternatives from the example consist of two normal actions with one parameter each.

Goals $g \in G$ is a goal which can be of the type maximize (`MAX!`), minimize (`MIN!`), fix (`FIX!`) or satisfy (`SAT!`). It measures a variable of the alternatives context according to its type. For example, the goal `FIX!(suggestion, user.interest)` states, that the value `suggestion` should be close to `user.interest`. In the above example the variable `model.fun` should be maximized. MADL goals are used by the decision rules to sort alternatives. (Not all types of goals may be used in every decision rule.) Each alternative $a_i \in A$ is assigned a score v_i, which ranks the alternative in respect to the others.

Constraints In addition to goals, MADL constraints eliminate alternatives from the result set, if they do not satisfy the constraint. For example `GOAL model.spendMoney < 1000` will filter out all alternatives which have a context with `costs` ≥ 1000.

Variables Variables change according to the context of the alternatives and are thus calculated by the PM. In the above example you can see the variables `model.spendMoney`, `model.money` and `model.fun`. Note that all variables in MADL are probability distributions. The example `foo = [30%: 1, 70%:2]` states that the value of `foo` is expected to be 1 in 30% and 2 in 70% of all cases.

Functions MADL functions allow altering of variables. As MADL's focus is on decisions, they cannot be implemented in MADL, but may be imported from a MADL model which can be implemented in Java. Note that MADL functions only take MADL variables, which are probability distributions, as arguments. The concept of risk (e.g. see [11]) can be modeled in MADL by integrating a (given or self created) risk function into the goal of the decision. An example for this is the last goal in the above decision.

Decision rules Currently, MADL supports the following decision rules:

Weighted Sum calculates the weighted sum. If no decision rule is denoted, the weighted sum is used.

Pareto Set returns the non dominated set of alternatives.

TakeTheBest identifies the best alternative according to *Gigerenzer et al.*'s TakeTheBest decision rule, which shall resemble human decision behaviour.

MiniMax compares the minimal goal attributes and decides for the alternative with the greatest goal attribute.

MaxiMin compares the maximum goal attributes and decides for the alternative with the smallest goal attribute.

Random takes a random alternative.

Count calculates v_i as the sum of identical alternatives. This is especially usefull when combining multiple strategies as discussed below.

User asks the user to select a set of best alternatives.

Other decision rules may be implemented in Java.

Now we will illustrate the usage of multiple decision rules. The following example shows the decision `routes` with three alternatives and three goals.

```
routes{
  ALT [WAY1, km = 90, costs = 0, jam = 10%]
  ALT [WAY2, km = 80, costs = 3, jam = 40%]
  ALT [WAY3, km = 80, costs = 4, jam = 30%]
  GOAL MIN!(km)
  GOAL MIN!(costs * 5)
  GOAL MIN!(jam * 50)
}
```

To use a decision rule other than `TakeTheBest`, one can inherit from the decision: `TakeTheBestDecision strategy1 EXTENDS routes {}`.

As there exists no best decision rule, *de Almeida Cunha* [13] suggests to minimize the effect of the different decision rules by applying multiple decision rules to the problem. This concept can be implemented in MADL, by selecting the alternative, which has been evaluated best by multiple decision rules and which is included in the Pareto optimum of all alternatives.

3.3 Mode of Operation

The mode of operation will be explained with an short example. The following code shows two decisions `D1` and `D2`.

```
CON test.aModel AS model

SumDecision D1 {
  ALT [a1, <D2]
  ALT [a2]
  GOAL MAX!(model.satisfaction)
  GOAL MIN!(model.costs)
}
SumDecision D2 {
  ALT [b1]
  ALT [b2]
  GOAL MAX!(model.x)
  GOAL MIN!(model.y)
}
```

The MADL program connects to some PM `test.aModel` Each decision contains two alternatives and two goals. Let us assume, one wants to evaluate `D1`. As the first alternative of `D1` imports the best alternatives of `D2`, `D2` has to be evaluated first. For each alternative of `D2` the result of applying the action to the model is calculated. As the variables are discrete probability distributions, all possible future contexts have to be calculated.

For a context of the first alternative with `x = [50% : 1, 50% 3]` and `y = [50% : 1, 50% 5]` we get $E(x_{b1}) = 2$ and $E(y_{b1}) = 3$. Assume $E(x_{b2}) = 2$ and $E(y_{b2}) = 4$. Based on the resulting contexts, a score v_i is calculated according to it's decision rule, using the expected value of the variables. As the weighted sum of both alternatives is equal, $E(x_{b1})-E(y_{b1}) = E(x_{b2})-E(y_{b2})$, both alternatives are imported to `D1`. This import can be seen as a new MADL decision:

```
SumDecision D1* {
   ALT [a1, b1]
   ALT [a1, b2]
   ALT [a2]
   GOAL MAX!(model.satisfaction)
   GOAL MIN!(model.costs)}
```

4 EVALUATION

Three different proof of concept use cases have been implemented in MADL to show its applicability. Decisions, which elements to show in printer dialogs have been implemented in MADL. They take into account whether the document to print contains colour, its length and the estimated costs of the print. Some constraints which only hold for specific alternatives were cumbersome to program. In further development of MADL, alternative specific constraints could be written behind the actions of the alternatives, e.g. `ALT [action] x > 3`, to ease this.

Also an assistant for a virtual digital video recorder has been programmed in MADL. It helps people to delete or compress previously recorded films from the disk, when there is no free space for the recording of an additional film. A user study with nine participants showed that using the MADL assistant, users needed 16.5% less mouse clicks. This experiment also indicates, using MADL you can isolate business logic from “user assistance logic” to foster independent development, testing and maintenance of each.

Route selection for a navigation system by using multiple strategies has also been successfully implemented. Furthermore, MADL was successfully integrated in a program to control a robot from our Chair of Artificial Intelligence and in a system to plan HCI dialogs.

5 CONCLUSION AND FUTURE WORK

For closing the gap between abstract goals and concrete adaptions in HCI, we invented the new programming language MADL. Therefore we analysed goals and decisions in HCI: Goals are various, contradictory and hard to measure. Decisions are deeply structured, diverse and include risk and uncertainty.

This led us to design an abstract programming language for multi-attributive decisions called MADL. It allows to model the hierarchy of decisions in HCI and make decisions with multiple goals with this programming language. MADL can handle risk and uncertainty and integrates different decision rules. The language can be extended with own prognosticating models, functions and decision rules and it is possible to integrate MADL in own software. We also showed the applicability of MADL in three different use cases.

In a further step we want to integrate MADL into a route generation algorithm based on the $h_\epsilon u$-heuristic [14]. There are still additions like hierarchical composition of goals, expansion of the hiding concept from actions to alternatives and goals, caching of contexts and others to implement. An interesting topic for future research would be to allow recursion by adding alternative specific constraints to gain a Turing-complete system.

References

1. Yoon, P.K., Hwang, C.L., Yoon, K.: Multiple Attribute Decision Making: An Introduction (Quantitative Applications in the Social Sciences). Sage Pubn Inc (March 1995)
2. Card, S., Moran, T.P., Newell, A.: The Psychology of Human Computer Interaction. Lawrence Erlbaum Associates, Hillsdale, NJ (1983)
3. Kleindorfer, P., Kunreuther, H., Schoemaker, P.: Decision Sciences: An Integrative Perspective. Cambridge Univ. Press, New York (1993)
4. Paterno, F., Mancini, C., Meniconi, S.: Concurtasktrees: A diagrammatic notation for specifying task models. In: INTERACT '97: Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction, London, UK, UK, Chapman & Hall, Ltd. (1997) 362–369
5. Johannsen, G.: Mensch-Maschine-Systeme. Springer, Berlin Heidelberg (1993)
6. Dix, A., Finlay, J., Abowd, G.D., Beale, R.: Human Computer Interaction. 3. edn. Pearson, Harlow, England (2003)
7. Shneiderman, B.: Designing the User Interface: Strategies for Effective Human-Computer Interaction. 3. edn. Addison Wesley Longman, Reading [u.a.] (1998)
8. van Welie, M.: Task-based User Interface Design. PhD thesis, Vrije Universiteit (April 2001)
9. Rothrock, L., Koubek, R., Fuchs, F., Haas, M., Salvendy, G.: Review and reappraisal of adaptive interfaces: toward biologically inspired paradigms. In: Thoretical Issues in Ergonomic Science. (2002) 47–84
10. Görz, G., Rollinger, C.R., Schneeberger, J., eds.: Handbuch der Künstlichen Intelligenz. 4. edn. Oldenbourg, München (2003)
11. Laux, H.: Entscheidungstheorie. 5th, improved edn. Springer, Berlin [u.a.] (2003)
12. Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press (1976)
13. de Almeida Cunha, C.J.C.: Ein Modell zur Unterstützung der Bewertung und Auswahl von Strategiealternativen. PhD thesis, Reinisch-Westfälische Technische Hochschule Aachen (January 1989)
14. Zenker, B., Ludwig, B.: Rose – an intelligent mobile assistant. Proceedings of the 2nd International Conference of Agents and Artificial Intelligence (2010)