Combining Finite State Machine and Decision-Making Tools for Adaptable Robot Behavior

Michalis Foukarakis[1], Asterios Leonidis[1],
Margherita Antona[1], Constantine Stephanidis[1,2]

[1]Foundation for Research and Technology – Hellas (FORTH)
Institute of Computer Science
Heraklion, Crete GR-70013, Greece
[2] University of Crete, Department of Computer Science, Greece

{foukas, leonidis, antona, cs}@ics.forth.gr

Abstract. Modeling robot behavior is a common task in robot software development. However, its difficulty grows exponentially along with system complexity. To facilitate the development of a modular, rather than monolithic, behavior system, proper software tools need to be introduced. This paper proposes combination of a well-known finite state machine and a custom decision-making tool for implementing adaptive robot behaviors. The notion of automatic behavior adaptation reflects the capability of the robot to adapt during runtime based on the individual end-user, as well as the particular context of use, therefore delivering the most appropriate interaction experience. While each tool on its own can be used towards that aim, a unified approach that combines them simplifies the task at hand and distinguishes the roles of designers and programmers. To demonstrate the methods' applicability, a concrete example of their combined use is presented.

Keywords: State Machine, Adaptation, Behavior, Decision-Making

1 Introduction

Typically, when designing and implementing complex robot behaviors, software developers have to cope with complicated systems with many components that need to efficiently cooperate. Several different technologies and tools can be used for this purpose. Probably the simplest, but the most frequently used one, is a finite state machine [1]. A Finite State Machine (FSM) includes an arbitrary number of states, at any given time only a single state is selected (current state), whereas a change among them is initiated by an event or a condition. They are used in many domains, enabling both hardware and software applications. This paper proposes the use of a tool that complements authoring of hierarchical state machines using an external decision-making mechanism for modeling and creating complex adaptive robot behaviors.

Other approaches towards robot behavior adaptation and decision-making include: (i) cognition-enabled robot control using reasoning mechanisms [2], (ii) generation of behavior networks using finite state machines [3], and (iii) the integration of behavior trees [4]. However, FSMs have been selected in this work both for their simplicity and for the fact that many programmers are already familiar with them and can maximize their potential [5-6]. Additionally, the proposed decision-making mechanism can be easily integrated to provide robust behavior structures and to negate some of FMSs' inherent drawbacks, such as limited non-reusable logic.

Both SMACH and DMSL are used in various components of the HOBBIT system [7]. SMACH is used in the development of scripted robot actions (e.g., locating the user), whereas DMSL is used for setting robot initialization parameters and decision-making for HOBBIT's fitness application. However, in the current implementation they are not combined.

Based on the experience acquired in the development of HOBBIT, this paper proposes the augmentation of ROS-compatible [8] finite state machines built using the SMACH library [9] with custom logic rules encoded in a language specifically designed for adaptation oriented decision-making.

2 Creating Robot Behavior with SMACH

The work presented here is based on the use of state machines for describing various robot tasks and actions. This can generally be achieved by mapping high-level tasks (e.g., navigating, planning, scanning the environment) to state machines, which in turn contain other states and state machines for more specific tasks. Building such a system is not trivial and requires careful design of states and state transitions, a robust method of selecting the correct transitions between them, as well as powerful and intuitive implementation tools.

SMACH is a python library that allows the rapid creation of complex robot behavior using state machine concepts. The prerequisite is that all possible states and transitions have to be known in advance and described explicitly, and the scheduling of the tasks has to be sufficiently structured. The advantages of using SMACH for describing robot tasks are many:

- The syntax is python-based, it is suitable for fast prototyping and allows designing complex hierarchical state machines.
- Different modules and tasks are described in a uniform way. State machine containers can describe high-level entities and states can describe actions. State transitions represent the changes in task flow.
- Any relevant data can be easily passed between states.
- State preemption is supported, which can be useful when tasks have to be aborted.
- An introspection tool is included, which doubles as a debugging interface for the state machine execution.
- Many functional states and state machine containers are included in the library and serve different purposes – for example a "Concurrence" state that enables many states to be executed in parallel.
- The library is fully compatible with ROS.

Fig. 1 shows an example state machine for a home robot which might need to locate the user in order to interact with him/her. The robot initially collects information about the current environment and from the user preferences or robot current status.

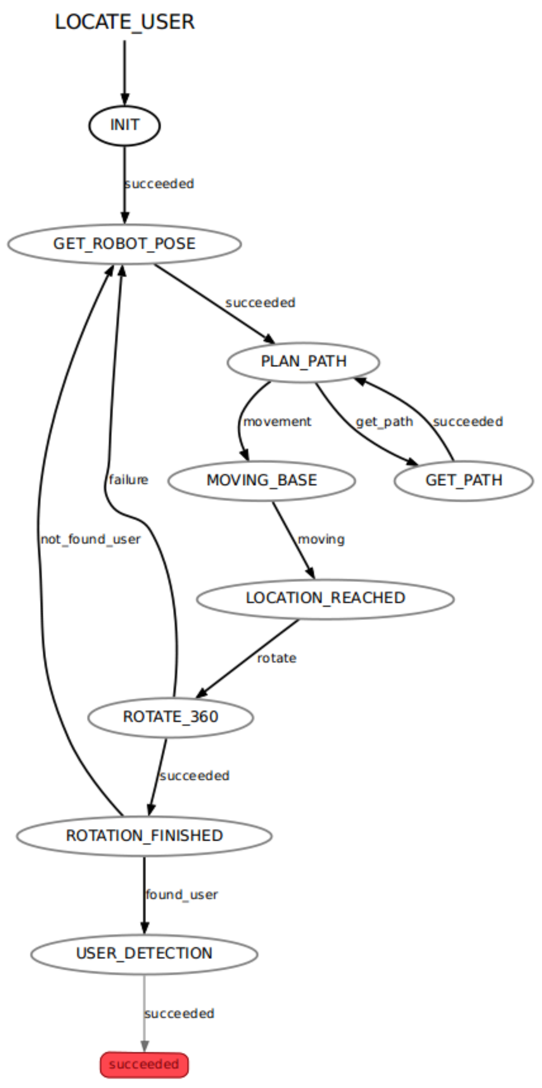

Fig. 1. State machine for the task "locate user". The error states have been omitted to save space.

This information includes current robot pose, last known user position, battery level, user's favorite room, etc. This is done inside states "INIT" and "GET_ROBOT_POSE". Then, the next step is to calculate the path to the next destination, which means that the execution logic transitions to state "PLAN_PATH".

From this point, the robot tries to move to each planned position and rotate in order to detect the user. These actions are described by their corresponding states.

Modeling robot behavior using state machines and state transitions is already a suitable method for implementing robot software. However, this can be improved upon by introducing a more sophisticated decision-making system that can not only select the appropriate transitions, but can also be used to make more complex decisions concerning behavior adaptation. The proposed decision-making mechanism is described in the next section.

3 Decision-Making Mechanism

In order to develop a complete behavior adaptation system and produce consistent behaviors for the robot, a decision-making mechanism can be used to model the different actions and choices that the robot is required to take according to the circumstances. This mechanism should be able to take into account different kinds of parameters that encapsulate the robot, user and environment status, and take the corresponding action according to the behavior rules that have been defined.

3.1 Decision-Making

The notion of automatic behavior adaptation reflects the capability of the robot to adapt during runtime based on the individual end-user, as well as the particular context of use, by delivering the most appropriate interaction experience. The storage location, origin and format of user-oriented information may vary. For example, information may be stored in profiles indexed by unique user identifiers, may be extracted from user-owned cards, may be entered by the user in an initial interaction session, or may be inferred by the system through continuous interaction monitoring and analysis. Additionally, usage-context information, e.g., user location, environment noise, etc., is normally provided by special-purpose equipment, like sensors, or system-level software. In order to support optimal robot behavior adaptation for individual user and usage-context attributes, it is required that for any given robot task or group of robot activities, the implementations of the alternative best-fit behavior actions are appropriately encapsulated.

During runtime, the robot software relies on the particular user, robot and environment profiles to adapt its behavior on the fly, doing the appropriate actions required for the particular end-user and usage-context. In the context of user interfaces, this type of best-fit automatic adaptation, called interface adaptability, was originally introduced in the context of adaptable and adaptive user interface development [10]. In the present work, runtime adaptation-oriented decision-making is engaged so as to select the most appropriate behavior action for the particular user, robot and environment profiles for each distinct part of the robot behavior.

The role of the decision-making component is to effectively drive the behavior adaptation process by deciding which actions or commands need to be selectively executed (or in other words, “activated”). The behavior adaptation process has inherent software engineering implications on the software organization model of the system components. More specifically, as for any component (i.e., part of the interface / behavior to support a user activity or task) alternative implemented incarnations may need to coexist, conditionally activated during runtime based on decision-making, the need to accommodate different behaviors arises. In other words, there is a need to organize interface components or robot actions around their particular task contexts, enabling them to be supported through multiple deliveries.

3.2 The Decision Making Specification Language

For implementing decision-making for robot behavior, the Decision Making Specification Language (DMSL) is proposed. DMSL has been extensively used in the past [11] for user interface adaptation. For the current purposes, DMSL provides a tool for creating complex decision blocks that evaluate decisions for robot behavior.

The main features of the DMSL language are summarized below:

- Supports localized decision blocks for each component.
- Built-in user and context decision parameters with runtime binding of values.
- Can trigger other decision blocks, supporting modular chain evaluations.
- Supports activation and cancellation commands for system components.
- Provides a method for automatic adaptation-design verification.

3.3 Outline of the DMSL language

The decision-making logic is defined in independent decision blocks, each uniquely associated to a particular component; at most one block per distinct component may be supplied. These components form networks of decision blocks that each correspond to a particular system component of the robot

The outcome of a decision session is an activation command, which is directly associated to the component of the initial decision block. For example, when returning to the user after executing a specific task, the robot navigation system could request a decision about which side to approach the user from (left, right, center). The request then enables a specific decision component, which takes into account the appropriate parameters and produces a decision for the navigation system to use. To simplify rule authoring and maintenance, the decision-making logic is defined in independent "if-then-else" decision blocks that add minimal overhead to the robot's execution time. The activate commands return a response that depends on profile attributes at the time of the request. The example in section 4 demonstrates the use of these concepts.

Decision parameters. The decision parameters are defined as a built-in object whose attributes are syntactically accessible in the form of named attributes. The binding of attribute names to attribute values is always performed at run time.

Each parameter belongs to one of the three profile types: user, robot and environment. This distinction is used for categorization purposes and does not impose any special semantic restrictions or characteristics. The supported parameter types are strings, numbers and true/false values. A description of each profile type follows:

- User profile: The user profile contains personal data associated to a specific user. It includes both static personal data and dynamic data gathered via activity monitoring. Such parameters include (but not limit to):
  - General information; i.e., username, name, surname, age, sex, nationality, preferred language, computer skill, etc.
  - Communication details; i.e., street address, city, postal code, country, e-mail, home telephone number, mobile telephone number, etc.
  - Preferences; i.e., look & feel preference, dialogue mode (guided, simple, normal, advanced), interaction style (touch, manual scanning, automatic scanning, etc.), auditory style, interaction preferences (only visual, only auditory, only gestural, both visual and auditory, etc.), distance during social and not social situations, frequency of spontaneous emotional expressions, frequency of suggestions concerning activities and exercises, frequency of robot initialized interactions, etc.
- Environment profile: The environment profile describes the robot's surroundings to ensure that it adapts its behavior to meet the user's goals within the given context of use. The environment profile includes:
  - Map of the environment used for navigation; i.e., obstacles, pathways, etc.
  - Map of the physical objects; i.e., table, chairs, shelf, etc.
  - Map of the available actuators; i.e., type, name, location, command, etc.
  - Information about the available AAL sensors; i.e., type, name, location and current value, etc.
  - Contextual information resulting from sensor data (e.g. estimated location of user, activity)
- Robot profile: The robot profile describes the internal information of the robot during task execution and social interaction. It defines the current appearance and behavior of the robot as a socially behaving agent at the side of the user. The robot profile includes:
  - General information; i.e., name given to the robot by user, physical dimensions, number and position of sensors, etc.
  - Internal information; i.e., current emotion (e.g., attentive, happy), personality preferred by user (e.g. butler, pet-like), number and status of sensors, energy level and status of battery, position and status of gripper, position and status of tray, etc.
  - Spatial information; i.e., current location, certainty of position, location name, indivisibility with user, direction of movement, direction of sight, etc.

These parameters, apart from settings and preferences, can also describe system statuses and temporary values. For example, the parameter ROBOT.VoiceVolume could have the numerical value that represents the robot voice volume level setting. Another example would be a parameter called ROBOT.CurrentTask, which would indicate what task the robot is executing at a specific time. Finally, sensor data can be very useful for adapting robot behavior, and can also be described by similar parameters.

DMSL "if-then-else" decision blocks include parameters which ultimately define the result of the decision process, influencing robot behavior. The next section describes a detailed example of the way DMSL is used for making a common decision for a home-care robot.

4 Adapting Robot Behavior with Decision-Making

The system considered here includes a robot capable of functioning autonomously and exhibiting variable behavior. Its functions can be divided into tasks and can be explicitly defined. To implement adaptable behavior, different implementations of relevant tasks are included. The objective here is to facilitate robot behavior coordination by providing easy to use tools that separate tasks for programmers and designers and reduce the overall implementation time.

The designers are encouraged to use well-known concepts (state machines) and tools (SMACH) in order to build the initial high-level task architecture for the robot. The process is straightforward and does not require them to dig into complicated code or decide the way to transition from one state to another. They are only concerned with building abstract states, state machines and the possible methods to connect them via state transitions.

The programmers on the other hand use the already designed state machines and their responsibility is to implement the functionalities of each state (for example rotating the robot platform to detect the user requires giving commands to both the platform wheels for the movement and the cameras for detecting). The decision-making procedure is tightly coupled with these actions, since for each action the robot can take (which can also mean transitioning to another state), a decision might be required. The programmers use DMSL decision blocks to encode the logic required by each state for transitioning to the next or for in-state decisions.

In **Fig. 2**, a simple diagram of how these tools cooperate with the robot's components is shown. At the core of the architecture, SMACH state machines and transitions are responsible for defining high-level tasks and communicate with both the low-level components and the user interface. In addition, the decision-making mechanism interacts with all the other components in different ways. Firstly, the SMACH states will usually require a decision to determine the next state transition. This can be provided by evaluating DMSL rules and getting the desired result back. Secondly, DMSL can provide best-fit user interface alternatives for the robot. Finally, low-level components of the robot may need to set certain profile parameters according to sensor data or system state. These parameters are used by the decision-making mechanism to evaluate DMSL rule patterns and drive the behavior adaptation process.

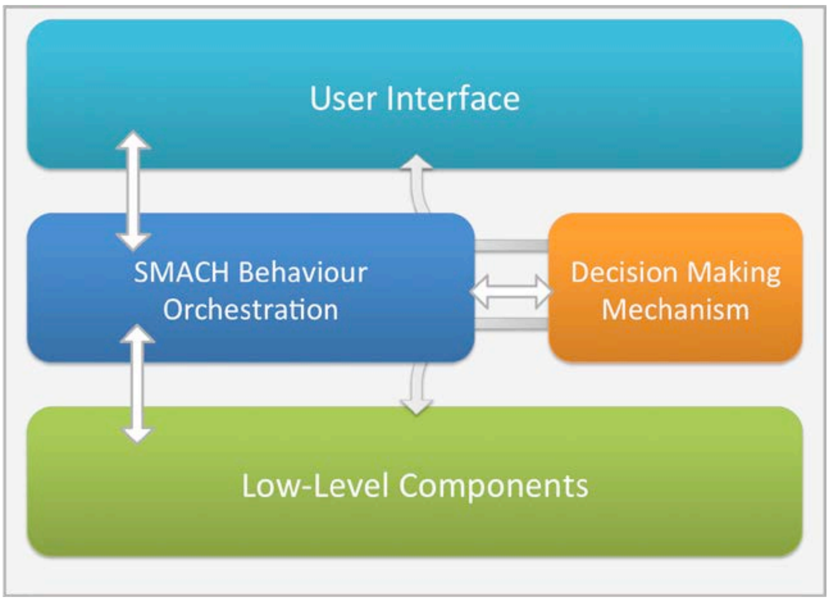

Fig. 2. Communication between SMACH, the decision-making mechanism and robot components

To demonstrate how to perform decision-making for an autonomous robot using the aforementioned procedures, a concrete example is presented of realistic robot behavior that can be found in a home-care robot: "explore the apartment in order to locate the user and ensure their safety". During movement, the robot is aware of its surroundings due to its internal localization and mapping. However, especially in home environments, objects change places and many obstacles can be found on the way to the robot's destination. In such a case, the robot has to decide how to overcome that obstacle. For the purposes of this example, it is assumed that two alternative options are available: (i) to call the user to remove the obstacle or (ii) plan an alternative route without bothering the user. The final decision is the based on many different parameters:

- The user might have limited mobility due to advanced age or any kind of disability
- The user might be facing partial or full hearing loss.
- The user might be preoccupied with another activity either unexpectedly or regularly at a given time every day; such cases can be determined through sensors or activity monitoring to identify user habits.
- The user might be unwilling to bond or interact with the robot.
- The user might not like to be disturbed at a certain time, or even at all.
- The robot might not be able to find another path to its destination.
- The robot might determine that the best alternative path is actually the longest one.
- The robot's battery level might be low.

- The robot is at the user's favorite room or the room the user is most likely in, given the time of day.

These are only examples of the possible variations on what can affect the decision to call the user or not. The first five parameters indicate that the robot should not disturb the user, while the rest suggest that the robot should call the user for help. **Fig. 3** diagrammatically represents the decision logic for this case.

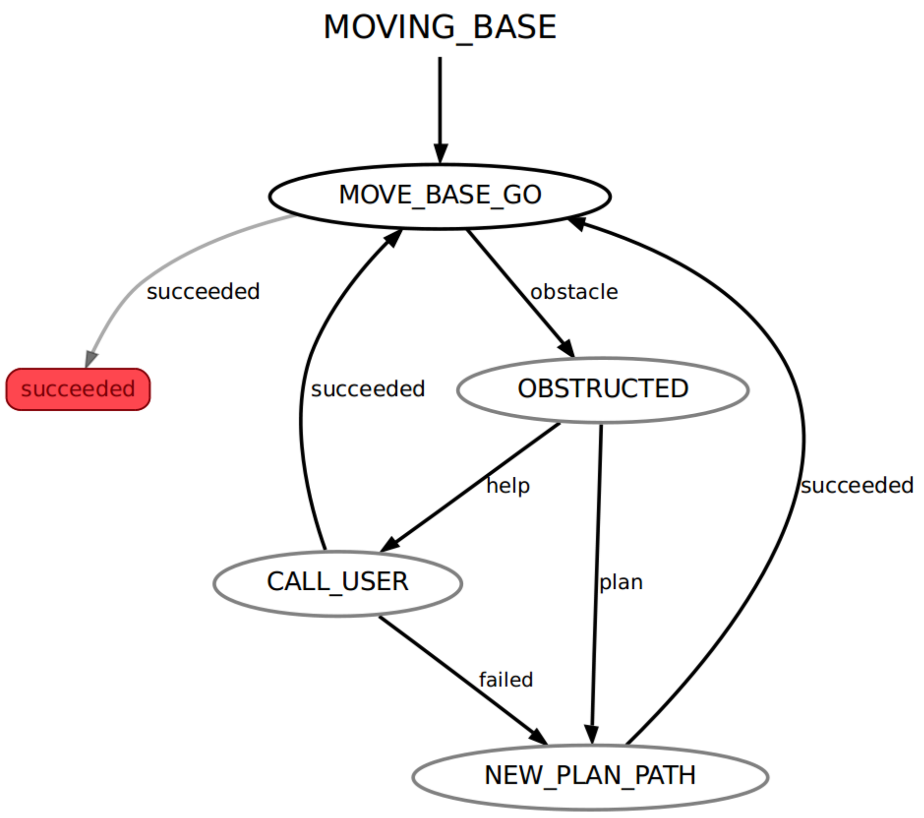

Fig. 3. State machine for robot movement. Error states have been omitted to save space.

The current state is "OBSTRUCTED" and the system needs to determine the next transition. The following DMSL rule is evaluated and the decision determines the next transition:

```
def neutralToBond {0,1,2}
def willingToBond {3,4,5}
def userFreeTimes {10, 11, 12, 13, 17, 18, 19, 20, 21 }

component ObstacleDetected [
  if params.user.hearing = false or
     params.user.age > 80 or
     not params.user.bondlevel in willingToBond or
     params.env.time not in userFreeTimes then
        activate "plan"
  else
   if params.robot.batterylevel < 10 or
      params.env.currentbestpath = false or
      params.robot.currentroom = params.user.favoriteroom
  then
        activate "help"
]
```

This rule includes some of the previously mentioned parameters and decides what the next state transition is. The if/else clauses determine which state transition is the one that best satisfies the current conditions, while the activate statements are used to return the result of evaluating this rule back to the state that requires it. In this case, the result corresponds exactly to one of the names of the two possible state transitions, "plan" and "help", as seen in **Fig. 3**.

In the same manner, more parameters can be used and more complex rules can be elaborated, according to the circumstances. Interestingly, a significant number of parameters used in one rule can be useful to another. Getting back to the state machine of **Fig. 1**, it can be seen that when the robot is trying to calculate the best path to take to search for the user, some of the same parameters can be reused. For example, the user's favorite room or the room the user is in usually at that time is a good indicator of where the robot should search first and it should influence its decisions at that point. Another point where such decisions could take place is when rotating to detect the user, to take into account if the user uses a wheelchair, is usually seated, etc.

5 Conclusion

This paper has described a finite state machine building tool and a decision-making mechanism for implementing complex robot behavior. It has been shown that these two tools complement each other well and can be combined without much effort to produce more advanced behavior systems.

Although the case reported here is restricted to specific parameters (home robot with wheels, user interaction and more), these tools can be used for many more applications that require executing robot tasks and using parameters to define the actions the robot takes.

Acknowledgements. Part of this work has been conducted in the context of the Project ICT-HOBBIT "HOBBIT The Mutual Care Robot", funded by the European Commission under the 7th Framework Programme (Grant Agreement 288146).

References

1. Kent, A., Williams, J. G. (Eds.): Encyclopedia of Computer Science and Technology (Vols. 25, Supp. 10). Applications of Artificial Intelligence to Agriculture and Natural Resource Management to Transaction Machine Architectures. CRC Press (1991)
2. Beetz, M., Jain, D., Mosenlechner, L., Tenorth, M., Kunze, L., Blodow, N., Pangercic, D.: Cognition-Enabled Autonomous Robot Control for the Realization of Home Chore Task Intelligence. Proceedings of the IEEE, vol.100, no.8, pp. 2454-2471 (2012)
3. Armbrust, C., Schmidt, D., Berns, K.: Generating Behaviour Networks from Finite-State Machines. In: Proceedings of the German Conference on Robotics (Robotik), Munich, Germany (2012)
4. Bagnell, J.A., Cavalcanti, F., Lei Cui, Galluzzo, T., Hebert, M., Kazemi, M., Klingensmith, M., Libby, J., Tian Yu Liu, Pollard, N., Pivtoraiko, M., Valois, J.-S., Zhu, R.: An Integrated System for Autonomous Robotics Manipulation. In: International Conference on Intelligent Robots and Systems (IROS), pp. 2955-2962 (2012)
5. Niemüller, T., Ferrein, A., Lakemeyer, G.: A Lua-based Behavior Engine for Controlling the Humanoid Robot Nao. In: Baltes, J., Lagoudakis, M.G., Naruse, T., Ghidary, S.S. (eds.) RoboCup 2009: Robot Soccer World Cup XIII. LNCS, vol. 5949, pp. 240–251. Springer, Heidelberg (2010)
6. Datta, C., Jayawardena, C., Kuo, I.H., MacDonald, B.A.: RoboStudio: A Visual Programming Environment for Rapid Authoring and Customization of Complex Services on a Personal Service Robot. In: International Conference on Intelligent Robots and Systems (IROS), pp. 2352-2357 (2012)
7. Vincze, M.: HOBBIT - Towards a robot for Aging Well. ICRA 2013, Karlsruhe, Workshop on Human Robot Interaction (HRI) for Assistance Robots and Industrial Robots, May 6-10 (2013)
8. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: An open-source robot operating system. In: Open-Source Software Workshop Int. Conf. Robotics and Automation, Kobe, Japan (2009)
9. Bohren, J., Cousins, S.: The SMACH High-Level Executive. Robotics & Automation Magazine, IEEE, vol.17, no.4, pp.18-20 (2010)
10. Stephanidis, C.: The Concept of Unified User Interfaces. In C. Stephanidis (Ed.), User Interfaces for All - Concepts, Methods, and Tools. Mahwah, NJ: Lawrence Erlbaum Associates, pp. 371-388 (2001)
11. Savidis, A., Antona, M., Stephanidis, C.: A Decision-Making Specification Language for Verifiable User-Interface Adaptation Logic. Int. J. Soft. Eng. Knowl. Eng. 15, 1063 (2005)