Automatic Assembly of Adaptive User-Interfaces via Dynamic Discovery and Deployment of Profile Providers, Decision Makers and Component Repositories

Effie Karuzaki

Institute of Computer Science, Foundation of Research and Technology Hellas (FORTH)
karuzaki@ics.forth.gr, as@ics.forth.gr

Abstract. In this thesis, we focus on automatic coarse-grained composition of adaptive user interfaces relying on task trees. To achieve this goal we collect multiple user and context profiles from independent providers, and supply consolidated profiles to independent decision makers (DMs) according to their own profile models. Each DM chooses the best-fit component for a given task according to the supplied user and context profile. To enable an unambiguous naming scheme for components across repositories we propose the notion of globally unique task identifiers. Selected components are downloaded from the repositories and are assembled into the finally delivered user interface.

1 Introduction & Related Work

The term "adaptation" in computer science refers to changing aspects of an interactive system to best fit individual users, based on their personal information and the context of use. This is very important nowadays as the ever-growing computer market targets an ever-widening set of users with different backgrounds, tastes and needs. Through the years, several architectures [3-6], methods and tools have been proposed for building adaptive user interfaces. In [1], an engineering paradigm to automate UI adaptation is discussed. User interface plasticity is defined in [2], along with an architecture supporting it. The later was evolved to include run-time adaptation [3] and served as a base for CAMELEON [4], a reference framework trying to cover every adaptation need. Although CAMELEON works well for large-scale systems, it's too painful to adopt in smaller ones as developers have to model every aspect of the user interface and its behavior in many levels. Other approaches shift adaptivity towards individual widgets [7-9], promoting a more fine-grained UI design philosophy. These are usually difficult to adopt for large systems, since they are based on small UI elements and developers have design the whole UI from scratch. Finally, frameworks exist that describe user interfaces using description Languages (UIDLs) and task modelling [10, 11]. These techniques suggest very detailed UI descriptions, thus they are painful for large-scale systems. More recent work has proposed adaptation through extensible and modular frameworks, a rather promising approach for future interaction systems (small and large) as it focuses on reusability. For example, MyUI [12] exploits a set of repositories of device profiles, individualization, adaptation and interaction patterns to

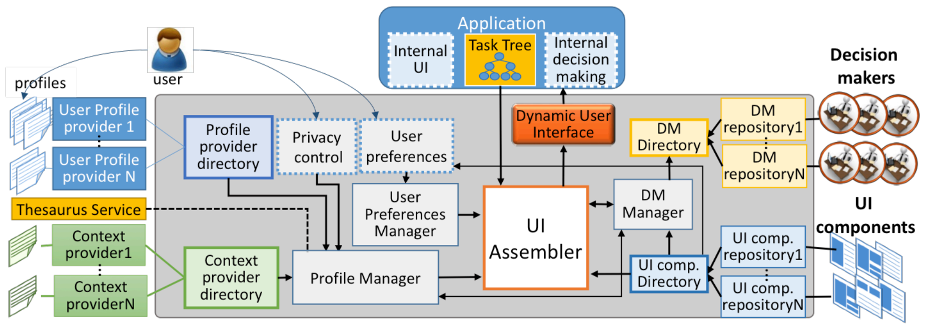

Fig. 1. The overall system architecture (gray box).

achieve UI adaptation during use, focusing on accessibility. MyUI supposes a common format for all design patterns and a common format among adaptation rules.

Collecting user and context information is an important part of the adaptation process. Towards this direction, many approaches have been proposed in the literature, utilizing common user and context models among the profile providers and the respective consumers. In the context of semantic web, the FOAF (Friend of a friend) ontology has been proposed for describing persons and their activities in a uniform manner. Although it has had limited adoption on the web[1], many propositions [13-15] are based on it for consolidating multiple user profiles. Virtual User Modelling and Simulation Standardisation project cluster [16] define a common vocabulary to avoid confusion among relative terms and user characteristics. Finally, [17] presents an approach for merging generic user and context profiles based on given priorities.

In our approach, application developers provide a coarse-grained task tree. The term coarse-grained is used to denote our focus on comprehensive dialogue components rather than on individual widgets. For the adaptive tasks, decision makers (DMs) select the most fitting UI components and we use them to assemble an adapted UI. DMs, Profile Providers and UI components can be developed externally and registered to our system repositories. The overall architecture is given in Fig.1, where the system backbone is depicted as a gray box. Notice that the application optionally has an internal UI in addition to the one assembled by our system. Additionally, an optional internal decision making mechanism is used to cover content adaptation needs.

2 Contributions

There are three main contributions that this dissertation aims to achieve:

- A methodology for collecting user and context profiles from independent distributed providers and assembling them into a unified user profile and a unified context profile that will be passed to independent distributed decision makers, without imposing model restrictions to the providers or DMs.

[1] http://en.wikipedia.org/wiki/FOAF_(ontology)

- A methodology for searching, rating and selecting the most qualified decision maker for each task, based on user and context profiles.
- The idea of adaptive UI composition from distributed UI components based on coarse-grained task trees, using distributed decision makers and profiles along with a system proposition that brings this idea to life.

3 Distributed Profile Management

Adaptation relies on user and context profiles. Today, user information can be retrieved from existing profiles in applications, social networks etc. The context of use can be retrieved from several services (e.g. gps) and sensors. We propose a mechanism for collecting and merging user and context profile information from distributed providers into a unified user profile and a unified context profile, while enabling providers to keep their own profile models and allowing user control over the retrieved information. No common models among the DMs or profile providers are supposed either. For example, a provider may refer to user hobbies as “user.hobbies”, another one as “user.interests”, while a DM may require it as “personal_info.free_time”. Each DM passes their model to the profile manager and the latter translates, transforms and merges the acquired profiles into unified ones that will conform to the DM’s model. A lexicon web service is used to provide synonyms for both model and profiles’ attributes. Synonyms in the model are then matched to synonyms in profiles, and common ones indicate matching of attributes, e.g. “interests” matches “hobbies”. Because attributes may have different meanings based on the profile structure, e.g. “name” can be found under “user.name” or “user.pet.name”, DMs should also provide a set of rules describing the alternative acceptable structures for each attribute that may be conflicted. The profile manager then uses the common synonyms along with the structure rules to produce a unified profile that matches the given DM model (Fig.2).

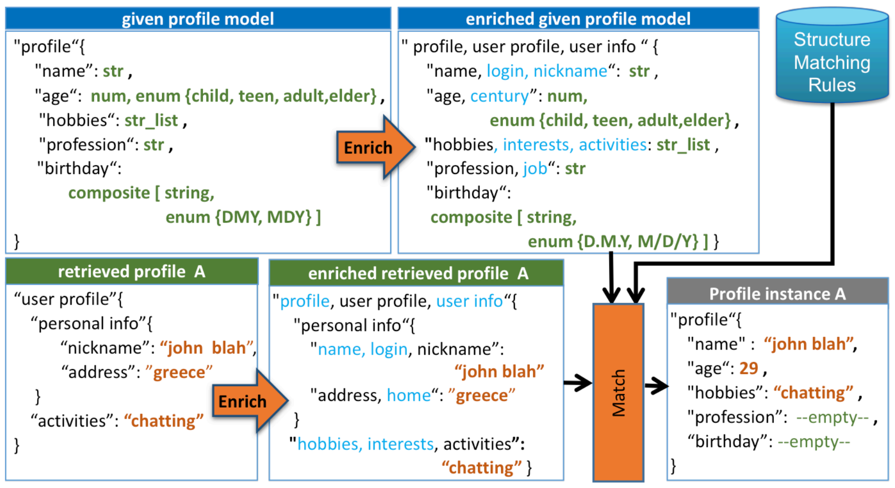

Fig. 2. Profile translation and transformation according to a given model

4 Task Model and Distributed Decision Making

Adaptive applications allow the realization of a task in alternative ways, depending on user and context profiles. We propose the assembly of adaptive user interfaces based on task trees, where one or more tasks can be realized via a user interface component retrieved from a repository. To avoid mismatches of tasks across components, we introduce the notion of globally unique task identifiers (GUTIDs). Tasks can be either adaptive, i.e. be realized through alternative UI components, or non-adaptive, i.e. bound to predefined components specified by the application developer. Adaptive tasks are assigned to a DM to find the UI component best matching the user and context needs. Thus, two challenges are raised: i) how to rate DMs, i.e. choose the DM that exploits most of the provided profiles while covering most of its model; and ii) how a DM can choose a UI component fitting the given application task tree. For the first challenge, a DM rating process is proposed (Fig.3). The idea is to rate the DMs based on two metrics, *coverage* and *utilization*. Utilization is produced per DM by the profile manager and refers to the percentage of the available profile information utilized by the given DM model. Coverage is computed by the given DM to reflect whether the information found in providers is enough for making good adaptation decisions. An equation provided by the application is then used to compute the final score. For the second challenge, a set of criteria must be met, first being the coverage of the given task. UI components have their own sub-task tree, which should match the application task tree and provide suitable hooks to allow further UI composition. Components are also expected to implement the API required by the application for the given tasks, ensuring their proper linkage to the application core. Finally, DMs contain rules expressing the adaptation logic for suggesting the component best fitting

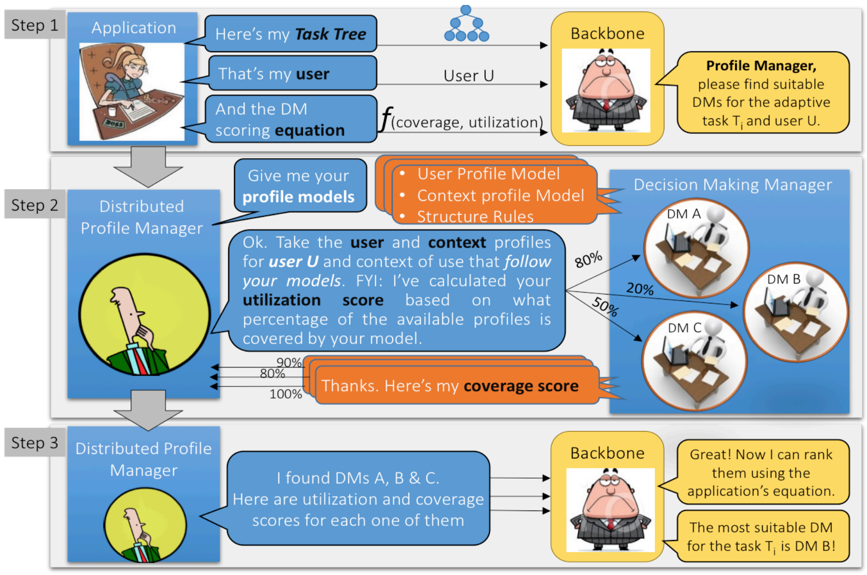
Step 1
Application
Here's my Task Tree
That's my user
User U
And the DM scoring equation
f(coverage, utilization)
Backbone
Profile Manager, please find suitable DMs for the adaptive task T_i and user U.
Step 2
Distributed Profile Manager
Give me your profile models
User Profile Model
Context profile Model
Structure Rules
Decision Making Manager
Ok. Take the user and context profiles for user U and context of use that follow your models. FYI: I've calculated your utilization score based on what percentage of the available profiles is covered by your model.
80%
20%
50%
DM A
DM B
DM C
90%
80%
100%
Thanks. Here's my coverage score
Step 3
Distributed Profile Manager
I found DMs A, B & C. Here are utilization and coverage scores for each one of them
Backbone
Great! Now I can rank them using the application's equation.
The most suitable DM for the task T_i is DM B!

Fig. 3. The process of choosing a suitable decision maker

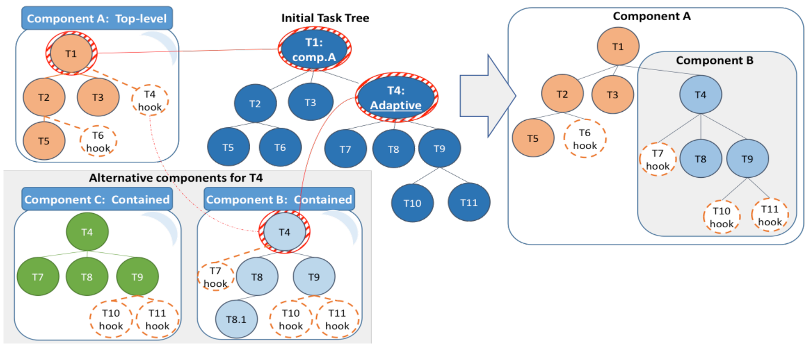

Fig. 4. An example of UI synthesis based on task trees.

the given profiles. No restrictions about the rule representation or the implementation language are posed; however, all DMs have to be registered in a directory, expose their capabilities and implement a common API to allow their uniform handling.

5 User Interface assembly

Once suitable components are selected for all adaptive tasks, we can proceed in assembling the final UI. In Fig.4, an example UI synthesis is sketched: T1 is a *non-adaptive* task, thus bound to a specific component (A). (A) must have hooks to enable other UI components to be attached under T1 and T2. Conversely, T4 is adaptive, thus can be covered by alternative UI components (B and C). Our system finds a suitable DM to pick the best fitting component for this task, say component B. Thus, comp. B is downloaded from its repository and linked to the corresponding hook (T4 hook) inside comp. A. Tasks 6, 7, 10 and 11 are not covered yet, so our system repeats the component-finding process until all application tasks correspond to a UI component.

UI components deliver parts of a user interface which can be reused by other apps to cover specific UI needs. Each UI component is tagged as top-level, contained or both, reflecting the type of the top-level container they use (in java it would be JFrame or JPanel). All components need to implement a common interface (API) to enable their uniform handling. Their implementation may be hand-written or derived from UI generator tools. It is part of this thesis is to provide directions to generators for producing components compatible with our system. Components should be registered in component repositories, carrying metadata describing the task(s) they deliver, the platform they target, the URL for downloading their binaries, etc. Repositories can be distributed as well, with some of them even being private. Finally, a directory service will provide information about which components reside in each repository.

6 Current progress and further research

The presented work is a thesis started three years ago (February 2011) and will be completed in the next 16 months. Currently, the entire system has been carefully designed and a profile manager for user profiles retrieved from social networks is developed. Algorithms for finding and ranking DMs and the syntax for describing matching tree structures have been sketched, and we have experimented with prototype test cases in Java (Desktop & Android) for UI synthesis. To this end, UI components (realized as jars) are downloaded from a URL and are assembled together into a final UI.

Further work remains to be done for the implementation of context profile utilization, decision making, dynamic component discovery and automatic UI assembly based on task trees, including matching component sub-task trees to the application task tree. Open research questions include ways in which the decision makers will refer to UI components, what properties each UI component will have and what information should a GUTID contain. Further research includes enabling run-time UI component substitution, ultimately targeting to dynamic and on-demand UI creation.

7 References

1. Savidis, A., Stephanidis, C., (2004) Unified user interface development: the software engineering of universally accessible interactions. UAIS (3-4) (pp 165-193) (2004)
2. Thevenin, D., Coutaz, J. (1999) Plasticity of User Interfaces: Framework and Research Agenda. Interact99, (pp. 110–117)
3. Calvary, et.al. (2003). A Unifying Reference Framework for Multi-Target User Interfaces, IWC 15(3) (pp. 289-308).
4. Calvary, G. et al(2002) The CAMELEON Reference Framework, Deliverable D1.1
5. Jaquero,V., Vanderdonckt, J., Montero, F., González,P. (2008). Towards an Extended Model of User Interface Adaptation: The ISATINE Framework. IFIP (pp. 374–392)
6. (1992)Arch: A Metamodel for the Runtime Architecture of An Interactive System, The UIMS Developers Workshop, SIGCHI Bulletin, 24(1), ACM Press
7. Calvary, G. et al (2004). Towards a new generation of widgets for supporting software plasticity: the “comet”. EHCI-DSVIS, (pp. 306-324)
8. Jabarin, B., T. C. Graham, N. (2003) Architectures for Widget-Level Plasticity. DSV-IS 2003.
9. Crease, M., Gray, P., Brewster, S., (2001). A Toolkit of Mechanism and Context Independent Widgets, Proceedings of DSVIS, (pp. 127-141)
10. Paterno, F., Mancini, C., Meniconi, S. (1997). ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models, INTERACT ’97 (Pages 362 – 369)
11. Pribeanu, C. (2005). An Approach to Task Modeling for User Interface Design. WEC(5) 5-8.
12. Peissner, M., Häbe, D., Janssen, D., Sellner, T. (2012). MyUI: Generating Accessible User Interfaces from Multimodal Design Patterns. *EICS ’12* (pp. 81-90).
13. Golbeck, J., & Rothstein, M. Linking social networks on the web with FOAF: a semantic web case study. In AAAI’08 (pp. 1138–1143). AAAI Press (2008).
14. Raad, E., Chbeir, R., & Dipanda, A. User Profile Matching in Social Networks. In NBiS’10 pp. 297–304, (2010)
15. Orlandi, F., Breslin, J., & Passant, A. Aggregated, interoperable and multi-domain user profiles for the social web. SEMANTICS ’12 p. 41 (2012).
16. Kaklanis, N, et al. An Interoperable and Inclusive User Modelling concept for Simulation and Adaptation. In Proceedings of the 20th UMAP (2012).
17. Bettini, C., & Riboni, D. Profile aggregation and policy evaluation for adaptive internet services. In MOBIQUITOUS 2004, pp. 290–298 (2014)