TRANSFORMING CONCUR TASK TREES MODEL INTO AN ABSTRACT USER INTERFACE

Margarita Atanasova,[1] Anna Malinova[2]

Abstract: Business applications are difficult to use for the average user. An adaptive user interface improves employees' productivity and is presented as a solution to this problem. However, developing user interfaces that are adapted to the needs and culture of the enterprise is time-consuming and expensive. We developed a software prototype for generating adaptive user interfaces that makes this process less time-consuming and more efficient. We propose an extension to the Cameleon Reference Framework project by Information Society Technologies, on the implementation level by adding an additional step for defining the Area of Business Operations. That way the prototype can extract business tasks for the selected industry therefore, presenting to the developer a more intelligent selection of predefined tasks. In this article, we also present a programming approach for transforming a task model, as defined by the ConcurTaskTrees notation, into an abstract user interface.

UDC Classification: 004.5; **DOI:** http://dx.doi.org/10.12955/cbup.v5.1067

Keywords: adaptive user interfaces, ConcurTaskTrees notation, abstract user interface, usability

Introduction

An adaptive user interface is defined as a user interface that is aware of the context of use and is capable of (automatic) changes according to the context (W3C Incubator Group, 2010). The context of use consists of the users, their tasks, the equipment (hardware, software, materials), as well as the physical and social environment in which the product is used according to ISO/IEC 25063:2014. One of the approaches for building adaptive user interfaces is model-based user interface development (Criado et al., 2010; Taktak et al., 2016; Akiki et al., 2016). In this approach models of high abstract level are defined which helps designers specify and analyze interactive software applications from a semantic point of view rather than focusing on the implementation. The aim is to increase the level of abstraction in a way that different adaptations could be easily applied on the different abstraction levels. The final goal is to produce a user interface that is capable of automatic change according to user skills, abilities, needs and environment (Pavlov, 2014; Pavlov et al., 2016).

The Cameleon Reference Framework (CRF), developed in a project by Information Society Technologies, defines four levels of abstraction in the UI development cycle (Calvary et al., 2003): Task and Domain, Abstract User Interface, the Concrete User Interface, and the Final User Interface. The final report from the W3C Incubator Group concerning this project is to start standardization in the area of Model-Based User Interfaces with the following work items: Unified Reference Framework for MBUI - A W3C recommendation that will formalize the CRF; Task, AUI and CUI meta-models' recommendations and other (W3C Incubator Group, 2010).

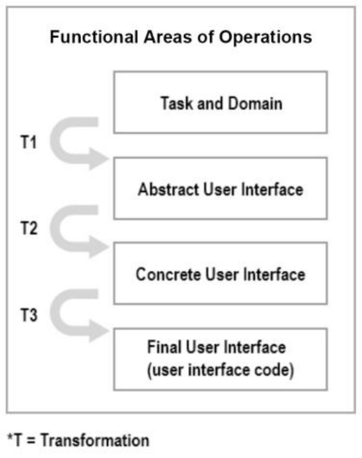
The first level in the development life cycle in the CRF is Tasks and Domain. In this step the logical activities that the users should perform to achieve their goals are defined. The next level is Abstract User Interface which is a formal platform-independent description of a user interface without details of the visual layout or behavior of the system (Constantine, 2003). The next level - Concrete User Interface is platform-dependent expression of the UI and defines more concretely what visual components should be used. The Final User Interfaces consists of the source code and it can be interpreted or compiled. CRF is used as a base for developing model-driven interactive systems (Blumendorf et al., 2006; Wu and Hua, 2013; Akiki et al., 2016).

We have developed a software prototype for generating adaptive user interfaces extending the Cameleon Reference Framework's four levels of UI development cycle. As the prototype is designed for building user interfaces for business information systems, we propose a modification to the CRF on the implementation level by adding an additional step for defining functional areas of operations (Figure 1). That way the prototype can extract business tasks for the selected industry, therefore presenting to the developer a more intelligent selection of predefined tasks. The UI

[1] Faculty of Mathematics and Informatics, University of Plovdiv Paisii Hilendarski, Bulgaria; margarita.hr.atanasova@gmail.com
[2] Faculty of Mathematics and Informatics, University of Plovdiv Paisii Hilendarski, Bulgaria; malinova@uni-plovdiv.bg

developers/designers can make a technology selection for the generated Final User Interface when they create their initial project.

Figure 1: Our extension of the four abstraction levels defined by the CRF



Source: Calvary et al. (2003)

In this article, we also describe how we implemented the transformation of the Task model into an Abstract User Interface. We developed a modern programming approach for that purpose.

Functional Areas of Operations

From a functional perspective, there are four types of information systems: Sales and Marketing, Manufacturing and Production, Finance and Accounting, Human Resources (Laudon and Laudon, 2015). Therefore, we implemented the selection of functional areas and business functions as a first step of the UI development process. We also developed the functionality of adding and deleting functional areas. We give the UI designers detailed descriptions for every business function as well as the ability to select more than one functional area. That way we enable them to easily build user interfaces for multifunctional information systems as CRM[3], ERP[4] and SCM[5].

ConcurTaskTrees Notation

In our software prototype, we have implemented sample task models using the CTT notation (Paternò, 2003; W3C, 2012) for different business tasks so that we ease and speed up the process of creating user interfaces for business applications. All these predefined tasks are stored in a library for quick access and reuse. The predefined tasks are limited on the second step by analyzing the selected functional areas from the user. We have also implemented the functionality of creating new tasks and saving them in a personal library for reuse.

This notation is specifically built for the aims of a model-based user interface development approach. Therefore, we chose it for base task notation in the task modeling step. ConcurTaskTrees notation focuses on the actions, has a hierarchical structure, has graphical syntax, which helps when describing different types of tasks and operators between them, supports object attributes and supports Task allocation. There are four task types in CTT: User Task, System Task, Interaction Task, and Abstract Task. CTT defines around 10 types of operators between tasks of the same hierarchical level.

Building ConcurTaskTrees Editor

Dealing with abstract user interfaces and task tree models requires an editor created to aim at maximum adaptability. Designers developing multiplatform adaptable user interfaces should be capable of doing it via a wider range of devices and operating systems (Rahnev and Stoeva, 2010) hence selecting a technology stack suitable for the Web seemed most reasonable. We built our own CTT editor as part of the software prototype, described in this paper, instead of using the CTT environment (Mori et al., 2002) because of several reasons: we wanted the task editor to have build-in

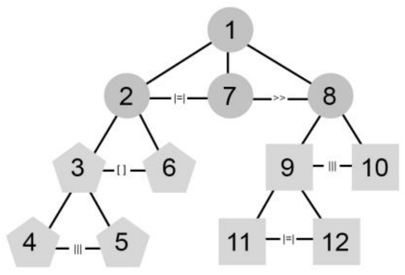
[3] CRM – Customer relationship management systems
[4] ERP – Enterprise resource planning systems
[5] SCM – Supply chain management systems

functionality in our prototype so that the UI developers do not need to install different products in order to create a user interface; the task modelling step needed to be connected to the first step (the selection of functional areas).

To use an open-source approach, we built the CTT editor described in this article using PHP, MySQL, JavaScript, HTML and CSS, and GoJS (Northwoods Software, 2017). The GoJS library is used for implementing the interactive diagrams for three of the five steps of the development cycle – Task Model, Abstract UI and Concrete UI. The engine for creating diagrams was chosen after a detailed research of several other libraries (e.g. GraphStream, Treant.js, Dracula Graph) that are used mainly for drawing graphs. GoJS works with specific JSON format, while the CTT environment generates different XML. If we were using this environment separately, instead of our own task editor, the UI developers should install this environment, build their own tasks, upload the generated XML into our system, then generate the rest of the UI model. If there were changes needed in the task model, they should have made them in the CTT environment and go again through all the steps that follow. With the CTT editor created in our web-based prototype, we save the users' time as they can traverse through all steps immediately with the only need to click the "Save" button to apply their changes in the models.

In the development process of the CTT editor several challenges arose: unclear and incomplete documentation of the graphing libraries; lack of examples; lack of opportunity to describe different node types and different types of the relations between them, which is a must-have feature for the selected notation as there are different types of tasks and links between them (Figure 2); none of the researched JavaScript libraries provided ready-to-use examples of hierarchical representation of graphs that have different types of connections between elements of the same level (siblings).

Figure 2: Simplified CTT notation model – different task types and relations between them.



Source: Authors

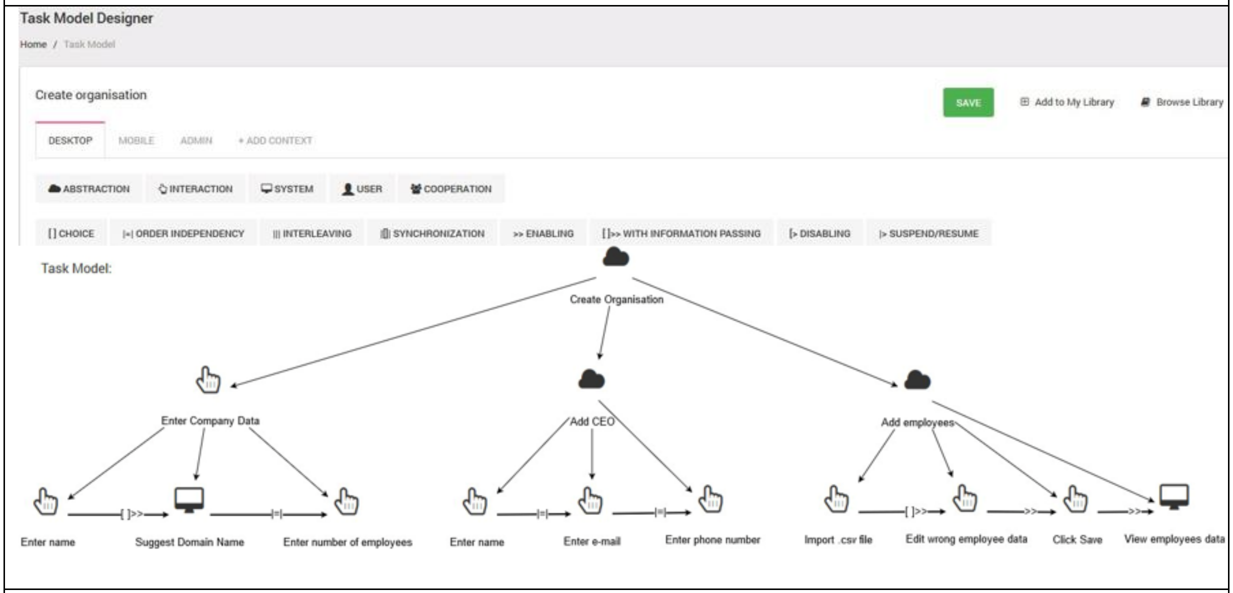
In Figure 2 a simplified CTT notation model is presented – this is what we need to build the task model data structure. The different node shapes represent different task types. The links between them have different symbols. This indicates the diversity of relations that two tasks can have. The structure should be a directed graph.

For our CTT editor we selected the GoJS library, which has several advantages: many different data structures can be described, multiple different relations between nodes can be described. It also renders HTML5 Canvas element or SVG without any server-side requirements. It is very convenient for building an integrated development environment as it has many advanced features for user interactivity such as drag-and-drop, copy-and-paste, in-place text editing, tooltips, context menus, templates, data binding and many other.

The development of the CTT editor went through several iterations of feature enrichment. First, we built a basic diagram with a tree structure. Then we added a toolbar with different types of tasks and relations. Functionality of adding and deleting tasks, children of the root element was created. We also created a graphical representation of different tasks: different icons according to task types plus labels under the tasks. Next we implemented the capability of renaming the task label as well as button "Save" to save the data structure in the database in JSON format with the task types and labels. Afterwards, we added the functionality of linking two sibling tasks with different types of relations (operators). In order to maintain horizontal structure between linked sibling tasks we defined a second layout for the linked siblings with a Horizontal link template. They ought to be in a horizontal line, not in vertical as placed by default by the graphs engine. Last, we implemented two-way data binding mechanism so that when the user makes edits on the JSON model and clicks the "Load" button, the changes are applied to the graph in the canvas and vice versa.

A result was achieved through which structures of tasks according to the CTT notation (Figure 3) can be successfully described, including: adding different types of tasks, using different icons and labels; the option of editing labels; connecting subtasks with different types of operators (links).

Figure 3: End result - CTT modeling editor. Example task model of Create Organisation task.



Source: Authors

One of the main difficulties during this module creation was linking tasks, while their hierarchical structure remains untouched. In other words, creating a representation of a multigraph where multiple edges are two or more edges that connect the same two vertices. A loop is an edge (directed or undirected) that connects a vertex to itself; it may be permitted or not, according to the application. In this context, an edge with two different ends is called a link. Two sibling nodes may have a directed link but this does not make the former node a parent of the latter. To achieve this, a new horizontal layout template had to be defined. This template was applied only to those elements that have relation to the same hierarchical level. In GoJS *isLayoutPositioned* ignores the diagram layout direction, which by implementation is vertical, representing a tree view. After that a TemplateMap object should be defined. This object is used to store templates from different object types – groups, links or node. In our case, we use two templates – the default one which supports vertical layout and horizontal.

In GoJS, there is a function model.toJson(), which transforms the model drawn in the HTML5 canvas into a JSON data structure. There is also a reversed function model.fromJson which takes the JSON and applies it to the model. An important detail here is that in order to keep the changes in the tasks names in the JSON file and vice versa, we added a two-way communication binding.

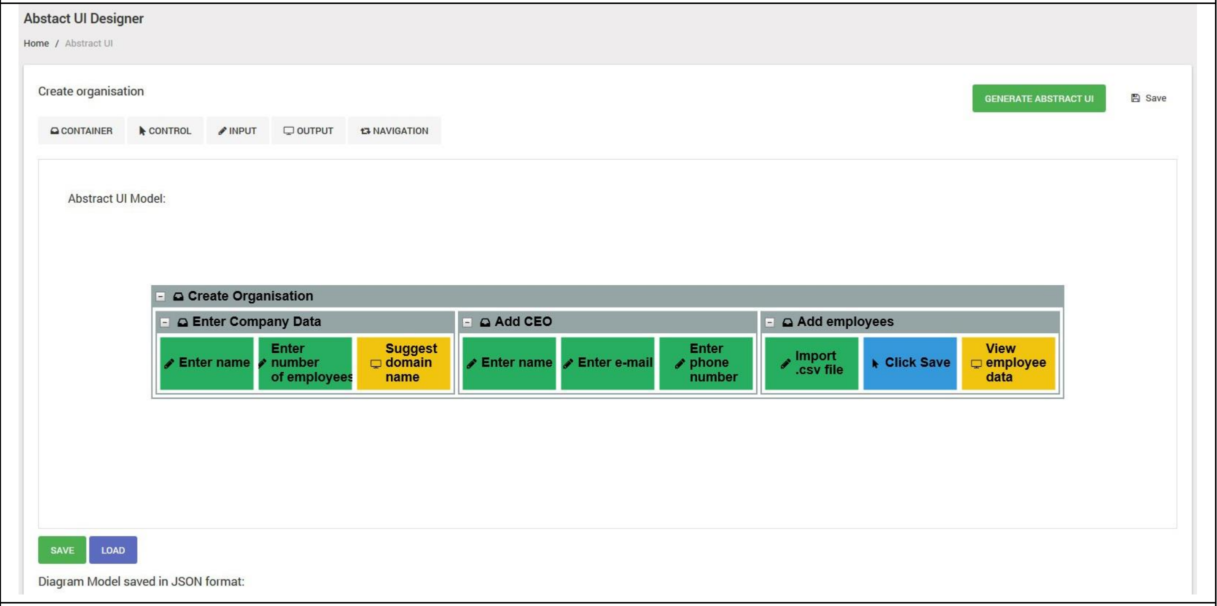
Transforming Task Model into an Abstract User Interface

In this section, we propose a programming approach for the process of automatic transformation of the task model into an abstract user interface model.

To build the structure of the abstract user interface, the topology of the tasks tree should be analyzed (Molina et al., 2012). First, every task that is a leaf is identified. These are the tasks that do not have any children. They will be the basic elements in the abstract user interface. Than we go up to the next level of the tree structure, which appears to be the container elements. They hold the basic elements. Then we go up in the tree structure using a recursive function and we define the other containers until we reach the root element which appears to be the main container.

Depending on the task type (abstract, interactive, system or user) and node type (parent or leaf), the task can be transformed into different abstract interactive objects (Montero and López-Jaquero, 2006). If the node is a parent it is always transformed into an Abstract Container. The only exception to the rule is if the task is a User type, because there is no abstract representation as this is a cognitive action performed by the user. The Abstract task type is always transformed into an Abstract Container as this task needs to be decomposed into simpler ones. If the Interactive task type is a leaf in the task tree structure, then it might be one of the four Abstract Interactive Objects - input, output, control or navigation. This choice is made by the user as this cannot be automatically selected. When the System task type is a leaf, it may be transformed into an output or navigation as this is a task performed by the software system. The result from the transformation of the task tree presented in Figure 3 is shown in Figure 4.

Figure 4: Generated Abstract User Interface of the Login task.



Source: Authors

Our process of automatic transformation of the task model into an abstract user interface model follows these steps:

- Fetch the CTT model in a tree structure.
- Convert the CTT tree structure recursively into a list structure.
- Iterate the list structure node by node to extract data needed for the AUI model.
- Populate the missing data with the user's provided input. Through the IDE the user has to select the type of the facets which cannot be explicitly determined by the transformation rules.
- Store all data in a list structure.
- Convert the final list structure to a format needed for rendering the AUI (JSON in the described case).

Conclusion

The creation of adaptive user interfaces benefits a lot from the model-based user interface development approach. That way the designers can focus more on the user tasks rather than the implementation. On the other side users are able to be part of the whole process as no code development is required when a proper development environment is used. Furthermore, designers can specify different adaptations for the different levels of abstraction. This paper details a programming algorithm for transforming the task model into an abstract user interface. This is part of a larger system that helps designers create context-aware user interfaces using a model-driven development approach. An evaluation study has to be conducted to evaluate the usability of the system and the development time of the final user interface. Future work includes adding a prior step of selecting a technology for building the final user interface so that the user is not only limited to HTML and CSS.

Acknowledgements

The work described in this paper has been partially supported by the project FP17-FMI-008 of the Scientific Fund of the University of Plovdiv "Paisii Hilendarski," Bulgaria.

References

Akiki, P. A., Bandara, A. K., & Yu, Y. (2016). Engineering Adaptive Model-Driven User Interfaces. IEEE Transactions on Software Engineering, 1118-1147.

Blumendorf, M., Feuerstack, S., & Albayrak, S. (2006). Event-based Synchronization of Model-Based Multimodal User Interfaces. Proc. Of Second International Workshop on Model Driven Development of Advanced User Interfaces. Genova: Springer.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. Interacting with computers 15 (3), 289-308.

Constantine, L. L. (2003). Canonical Abstract Prototypes for Abstract Visual and Interaction Design. International Workshop on Design, Specification, and Verification of Interactive Systems (1-15). Funchal, Madeira Island: Springer.

Criado, J., Vicente-Chicote, C., Padilla, N., & Iribarne, L. (2010). A Model-Driven Approach to Graphical User Interface Runtime Adaptation. 5th Workshop on Models@run.time at MODELS 2010, (49-59). Oslo.

Laudon, K. C., & Laudon, J. P. (2015). Management Information Systems: Managing the Digital Firm . Boston: Pearson.

Molina, A. I., Giraldo, W. J., Gallardo, J., Redondo, M. A., Ortega, M., & GarcíA, G. (2012). CIAT-GUI: A MDE-compliant environment for developing Graphical User Interfaces of information systems. Advances in Engineering Software, 10-29.

Montero, F., & López-Jaquero, V. (2006). IDEALXML: AN INTERACTION DESIGN TOOL A Task-Based Approach to User Interfaces Design. Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces, 245-252.

Mori, G., Paterno, F., & Santoro, C. (2002). CTTE: support for developing and analyzing task models for interactive system design. IEEE Computer Society, 797 - 813.

Northwoods Software. (2017). GoJS - Interactive JavaScript Diagrams in HTML. https://gojs.net/: https://gojs.net/

Paternò, F. (2003). ConcurTaskTrees: An Engineered Notation for Task Models. От D. Diaper, & N. A. Stanton, The Handbook of Task Analysis for Human-Computer Interaction (483-503). Mahwah, New Jersey: Lawrence Erlbaum Associates, Inc.

Pavlov N. (2014), User Interface for People with Autism Spectrum Disorders, Journal of Software Engineering and Applications (JSEA), Vol. 7, No. 2, ISSN: 1945-3116, pp. 128-134.

Pavlov N., Rahnev A., Mirchev K., Gardjeva T., Krushkova D. (2016), Responsive User Interface for People with ASD, International Journal of Pure and Applied Mathematics - IJPAM , Vol 111, No 1, pp. 105-119, ISSN 1311-8080 (printed version), ISSN 1314-3395 (on-line version).

Rahnev, A., & Stoeva, M. (2010). Principles and technologies for development of user interfaces for Web and desktop applications. National scientific conference "Education in the Information society", (308–317). Plovdiv.

Taktak, H., Riahi, I., & Moussa, F. (2016). A Model Driven Approach For Adaptive User Interfaces Specification: User, Task And Environment Impact. ACHI 2016 : The Ninth International Conference on Advances in Computer-Human Interactions (225-232). VENEZIA : IARIA.

W3C . (2012). Concur Task Trees (CTT) W3C Working Group Submission 2 February 2012. https://www.w3.org/: https://www.w3.org/2012/02/ctt/

W3C Incubator Group. (2010). Model-Based UI XG Final Report. W3C Incubator Group.

Wu, H., & Hua, Q. (2013). A Model-Driven Interactive System. International Conference on Information Computing and Applications 2013 (430-439). Singapore: Springer, Berlin, Heidelberg.