Web Tool for Goal Modelling and Statechart Derivation

João Pimentel, Jéssyka Vilela, Jaelson Castro
Universidade Federal de Pernambuco
Recife, Brazil
{jhcp, jffv, jbc}@cin.ufpe.br

***Abstract*—Creating and maintaining visual models is a time-consuming step in software engineering processes. In order to support the creation of some of these models, we have developed the Goal to Architecture tool (GATO). This web tool handles the creation and edition of goal models, as well as the derivation of statecharts. The particular variation of goal modelling supported by this tool contains four views: requirements view, design view, delegation view, and behavioural view.**

***Index Terms*—Goal modelling tool, Web tool, Statechart Derivation, Requirements-driven software adaptation.**

I. Introduction

Requirements engineering and architectural design, while addressing the system specification at different abstraction levels, comprise intertwined activities. The former focuses on the problem at hand, whereas the latter provides solutions for that problem [1]. One of the challenges in Software Engineering is to determine whether a system specification is actually a solution to the given requirements. We propose to tackle this challenge by systematically deriving the latter from the former. In earlier work we proposed a process for deriving architectural models from requirements [2][3]. However, these proposals handle only the structural view (e.g., components-and-connectors) of the architectural design, lacking any behavioural specification of the system-to-be. The behavioural view (e.g., statecharts) accounts for the order of execution of tasks that fulfils requirements, including concurrency and time dependencies for the system-to-be [4], allowing further refinement of the system and supports reasoning to detect potential deadlocks, termination and efficient use of scarce resources.

In recent work [5][6] we have proposed the MULAS framework for designing adaptive systems. This framework includes a systematic process for deriving behavioural models – expressed as statecharts [7] – from goal models through a series of refinements that introduce into the design: (i) *design tasks, design constraints* and *design assumptions*, (ii) *assignments* of responsibility, and (iii) *behavioural refinements*. *Design tasks* allow the architect to define tasks that, although not relevant for the stakeholders – thus not expressed in the requirements – are important in defining the behaviour of the system (e.g., *connect to server*, *log in*, and *check connection speed*). *Design constraints* refine requirements quality constraints into more concrete constraints (e.g., *use 128-bits encryption* may be a refinement of a *security* requirement). *Design assumptions* represent situations assumed during system design, often allowing some simplification of the system (e.g., *no more than 100 users will access the system simultaneously*). *Assignments* lead to an initial structuring of the system, in accordance with who/what (e.g., human actors, organizations, software components, etc.) is responsible for performing its different tasks. Lastly, *behavioural refinements* define the execution flows of the system and open the door for the transformation of requirements into statecharts.

Considering the effort required for creating and maintaining the different models of the process, we identified the need for creating a supporting tool. This *Goal to Architecture* tool (GATO)[1] is described in the following sections.

II. Requirements

The requirements for our supporting tool are presented in Fig. 1. The functional requirements were elicited based on what is necessary for supporting the enactment of the process proposed on [5][6]. The quality constraints were elicited from the needs of the authors when conducting and reporting case studies. It supports the edition of requirements elements (goals, tasks, domain assumptions, and quality constraints), as well as of the edition of design elements (design tasks, design constraints, design assumption, assignments, and behavioural refinements) and the edition of adaptation elements (awareness requirements and parameters). After included in the model, each element may be moved, renamed, or deleted.

Besides editing and managing goal models, it is also possible to create adaptation specifications (adaptation strategies and parameters) and to specify transitions (events and conditions). The tool also supports the derivation of statecharts from the design goal model. Lastly, the functionality of *Export to Zanshin* was included in order to facilitate the integration with a software adaptation component [8].

The GATO tool has three main quality constraints: *Portability* of model editing, *Reliability* of model managing, and *High Quality* of saved images. By portability we mean the ability to run over different operating systems: Windows, Mac, and Linux. Reliability, which is too abstract, was made concrete with an *Autosave* task. I.e., if the tool presents an autosave functionality, the system is considered to be sufficiently reliable. Lastly, in order to satisfy the *High Quality* constraint, it was decided to *Provide Vectorial Format*, such as SVG (Scalable Vector Graphics).

[1] Available at http://www.cin.ufpe.br/~ler/supplement/re2015/

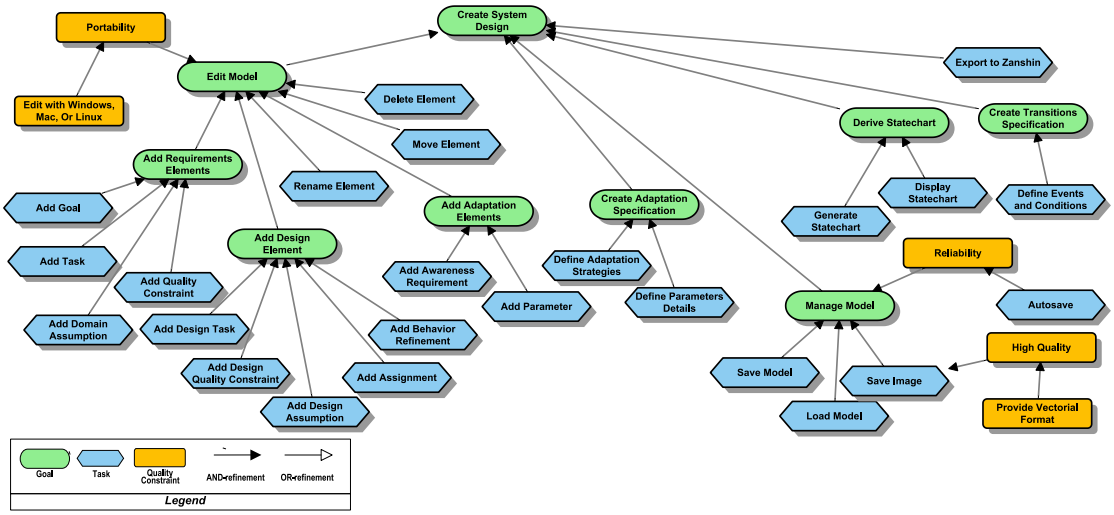

Fig. 1. Requirements of the Goal to Architecture tool (GATO)

III. Design and Implementation

The tool client was developed with web-based technologies, taking advantage of recent advances on web browser performance. A diagram is a SVG image that is rendered natively by web browsers. The creation and updating of said image was developed using the JointJS[2] library. The user interface of the client was developed using the Bootstrap framework[3] and the jQuery[4] library. The goal model is displayed with four different views, which facilitates its visualization. This feature was inspired by the STS-Tool [9].

On the server side, the module for deriving statecharts from flow expressions was built using the SableCC tool[5] , which automatically generates Java code for parsing an input text, as well as for creating and traversing an abstract syntax tree of the parsed text. The code is generated by SableCC taking as input a custom-defined grammar, which specifies the tokens and production rules of the language that will be parsed.

The code for actually identifying the states, transitions and concurrent states that should be derived from a given flow expression was developed on top of the depth-first tree traverser generated with SableCC. This mapper was wrapped up as a restful service using Jersey[6], allowing it to be invoked from our web-based client. The result of the derivation is then displayed back in the web client.

The use of a grammar-based approach for performing the goal model – statechart transformation was selected in detriment of model transformation languages such as ATL (ATL Transformation Language) and QVT (Query / View / Transformation), since the most relevant information for this transformation is the flow expression, which is textual. In order to verify the implementation of this algorithm we developed automatic tests comparing the results for given expressions to their expected results.

The result of the derivation is displayed back in two ways: as a list of states, their hierarchy, and transitions; and as a visual diagram. The creation of visual diagrams, from this output, is still in early development. Currently, the tool generates a statechart only with the atomic states; in future developments we expect to be able to include the super-states on the diagram.

Acknowledgements

Work supported by the ERC advanced grant 267856 “Lucretius: Foundations for Software Evolution”; as well as by the following Brazilian institutions: CAPES, CNPQ, and FACEPE.

References

[1] B. Nuseibeh, “Weaving together requirements and architectures,” Computer (Long. Beach. Calif)., vol. 34, no. 3, pp. 115–119, Mar. 2001.

[2] J. Castro, M. Lucena, C. Silva, F. Alencar, E. Santos, and J. Pimentel, “Changing attitudes towards the generation of architectural models,” Journal of Systems and Software, vol. 85, pp. 463–479, Mar. 2012.

[3] J. Pimentel, M. Lucena, J. Castro, C. Silva, E. Santos, and F. Alencar, “Deriving software architectural models from requirements models for adaptive systems: the STREAM-A approach,” Requirements Engineering, vol. 17, pp. 259–281, June 2012.

[4] F. Bachmann, L. Bass, P. Clements, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, “Documenting Software Architecture : Documenting Behavior,” Tech. Rep. January, 2002

[5] J. Pimentel, “Systematic Design of Adaptive Systems — A Control-Based Framework,” Ph.D. thesis, Universidade Federal de Pernambuco, 2015. Available at http://www.cin.ufpe.br/~ler/supplement/re2015/

[6] J. Pimentel, J. Castro, J. Mylopoulos, K. Angelopoulos, and V. E. S. Souza, “From Requirements to Statecharts via Design Refinement,” Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC 2014, pp. 995-1000, 2014.

[7] D. Harel, “Statecharts: A visual formalism for complex systems,” Science of computer programming, vol. 8, no. 3, pp. 231–274, 1987.

[8] V. Souza, “Requirements-based software system adaptation”, Ph.D. thesis, University of Trento, 2012.

[9] E. Paja, F. Dalpiaz, M. Poggianella, P. Roberti, and P. Giorgini, “STS-Tool: Specifying and Reasoning over Socio-Technical Security Requirements”, 6th International i* Workshop (iStar'13), 2013, pages 131–133, Valencia, Spain.

[2] Javascript library, available at http://www.jointjs.com/
[3] Available at http://getbootstrap.com/2.3.2/
[4] Javascript library, available at https://jquery.com/
[5] Available at http://sablecc.org
[6] Java library, available at http://jersey.java.net