Domain-Specific Modelling Language for Navigation Applications on S60 Mobile Phones

Janne Merilinna

VTT Technical Research Centre of Finland
P.O. Box 1000, 02044 Espoo, Finland
janne.merilinna@vtt.fi

Abstract

Domain-Specific Modelling Languages (DSML) provide an opportunity to have end-users at the centre of the software development process. Although end-users are seldom software developers, providing a language that both the end-users and software developers understand enables fluent communication between the stakeholders. In this paper, work in progress in the development of a DSML for navigation applications on positioning enabled S60 mobile phones is presented. The presented language enables the end-user to instantly experience the impact of changes in the models, by utilising a code generator that produces complete applications from the models. The architecture of both the supporting software framework and the generated applications are also discussed.

Categories and Subject Descriptors D.1.7 [**Programming Techniques**]: Visual Programming

General Terms Languages, Experimentation, Human Factors

Keywords code generation, end-user driven development

1. Introduction

Obtaining specifications for a software product directly from the end-users is worthwhile. In an optimal case, this enables the transforming of the end-users' will directly to a product. However, the end-users are seldom software developers themselves, thus the lack of a common language between the software developers and the end-users may become a barrier.

Domain-Specific Modelling Languages (DSML) can enable a fluent communication between the software developers and the end-users by providing a language easy enough to learn and understand [1]. This is achievable by providing a language that utilizes elements existing in the problem domain, instead of elements of a solution-space, which can be a stumbling block when using general-purpose modelling languages. By doing so, the end-users are already familiar with the language concepts, thus the learning curve is not too steep.

With code generation being a central process in the Domain-Specific Modelling (DSM), the possibility of transforming the models directly into a working application is feasible [1]. This enables end-users to instantly see the results of the modelling, thus enabling active participation in the requirements gathering and prototyping phases or even developing the software alone.

In this paper, we approach the end-user driven development with an experiment of developing a navigation applications product family [2] architecture for positioning-enabled S60 [3] mobile phones such as Nokia N95 [4]. The products of the family are not solely restricted to be composed of features selectable from a predefined list since a modelling language dedicated for the modelling of innovative navigation applications is also provided. The modelling language is supported by a code generator that generates complete code from the models.

The language is striven to be developed in such a way that even enables non-programmers, who would not otherwise be able to develop applications, to do so with the provided language. With the language, modellers are able to develop innovative navigation applications by utilizing map data provided by the OpenStreetMap[1] (OSM) [5] and are able to navigate both outdoors with GPS and indoors with the Database Correlation Method (DCM) [6].

This paper is structured as follows. First, the developed modelling language is illustrated by presenting a simple navigation application. Second, the implementation of the supporting software framework and the application architecture is presented. A discussion and conclusions close the paper.

2. Illustration of the DSML for Navigation Applications on S60 Mobile Phones

Fundamentally, navigation applications can be considered as applications that utilise positioning sensors, such as GPS, in order to show the location of the user on a map. Nevertheless, it is also common to have at least the following features:

- Zooming and panning of the map,
- Navigation, i.e. routing from source to destination, by utilising various routing criteria, and
- Browsing Point of Interests (POI), e.g. searching for the nearest bars, restaurants etc.

The other more advanced features can be considered to be composed from the above mentioned features.

Next in this section, the modelling language for navigation applications on positioning enabled S60 mobile phones (DSML for NavApp) is introduced by presenting an example application modelled with the developed language. Due to space limitations, not all of the language concepts can be presented here, but the example application should enable one to have an idea of the

[1] available under Creative Commons Attribution-ShareAlike 2.0

modelling language. Metaedit+ from Metacase [7] is utilized as a modelling environment.

2.1 Relation to the Existing S60 Language

Metaedit+ provides a set of example languages such as DSML for S60 [1, pp. 160-185]. This language includes a subset of elements provided by the Python for S60 (PyS60) [8]. The language enables the modelling of S60 applications almost in the WYSIWYG principle, thus providing a good starting point for the development of the DSML for NavApp.

Figure 1 represents the relation between the DSML for S60, DSML for NavApp, PyS60 and the S60 framework. As depicted, the DSML for NavApp also includes concepts of the existing language with additional concepts of its own.

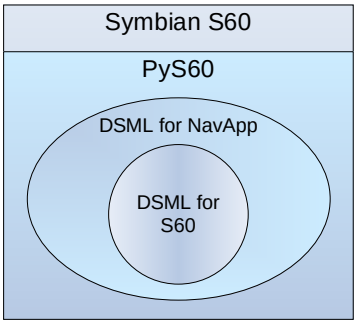

Figure 1. Relationships of the languages and frameworks.

2.2 A Simple Navigation Application as an Example

A simple navigation application is presented as an illustration of using the language. Figure 2 represents what will occur when the application starts. In this case, a pop-up dialog is shown first with three options. The application closes by choosing "Exit with the left sofkey, i.e. a button for accessing context-sensitive menus appearing at the bottom left of the screen of the mobile phone. If "About" is chosen, a note is shown and, as default behaviour after the note has disappeared, the application returns to the pop-up state. If "NavApp" is chosen, a navigation application called "NavApp" is launched. When the "NavApp" is closed, as default behaviour, the application returns to the pop-up state.

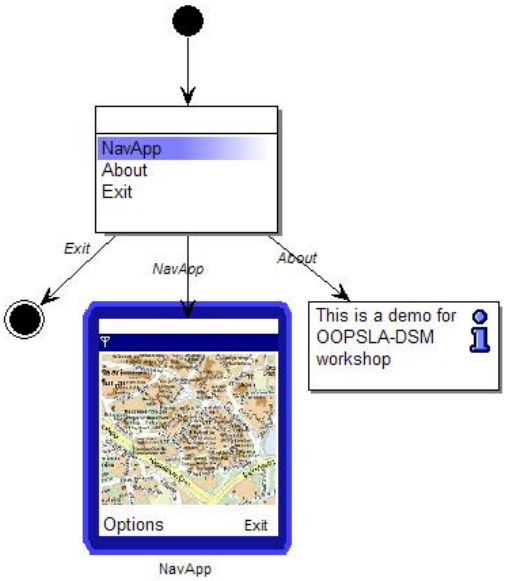

Figure 2. NavApp state machine.

Figure 3 represents the sub state machine of the NavApp, which is accessed from the NavApp entity in Figure 2 by decomposition. First, the menus and keyboard are defined with a "Menus" entity that provides a sub diagram where the definitions take place (see Figure 4). After the definitions, the application enters a loop where it is polled if the user is at the destination location. The destination can be defined by accessing one of the menus defined in Figure 4. When at the destination, a note is displayed to inform the user. The positioning ends after the note disappears.

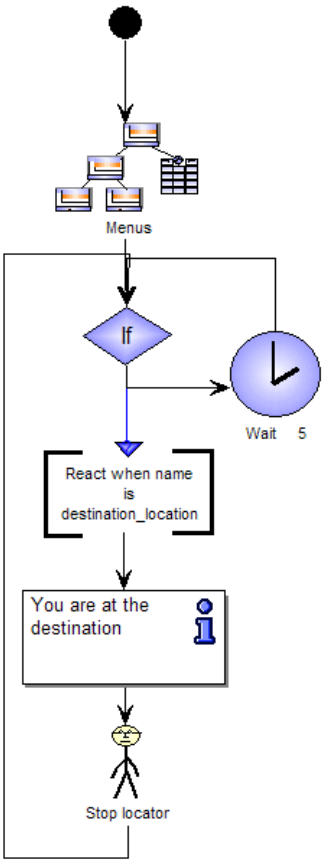

Figure 3. NavApp sub state machine.

Figure 4 represents definitions of the menus and the keyboard. First, zooming is attached to the <*+> and <#> keys followed by select menu definitions. By pressing the left softkey, a menu structure can be displayed where there is one parent node called “Navigate” having two child nodes, i.e. “Navigate to destination”, and “Stop navigating”. By choosing the latter, the navigation ends. By choosing the first mentioned, the user can type the name of the place where to navigate. After that, the route to the destination is computed from the current location and the navigation is started.

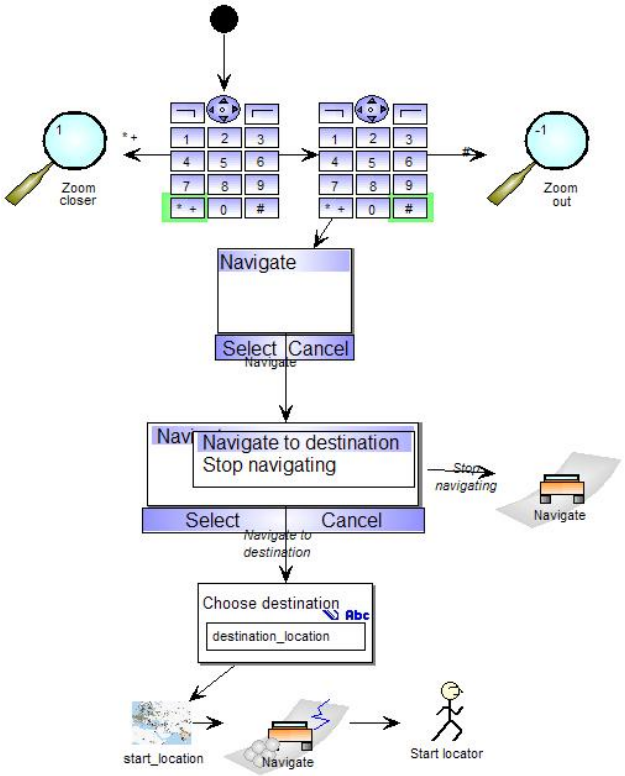

Figure 4. Menu and keyboard action definitions.

After the modelling, the code generator takes the models as an input and generates a Python source code on top of the supporting software framework. The generated application can then be installed into the phone. No additional manual source code writing is required.

3. Implementation of the NavApp

3.1 Architecture of the NavApp Framework

The basic architecture of the DSM can be considered to consist of three layers which are the software framework, the code generator and the metamodel. As illustrated in Figure 5, the metamodel provides all the rules on how to model applications. The code generator is responsible for taking modelled applications as an input and generates code from the models. It is common to have a software framework on top of the target platform, in order to make the code generation easier. [1]

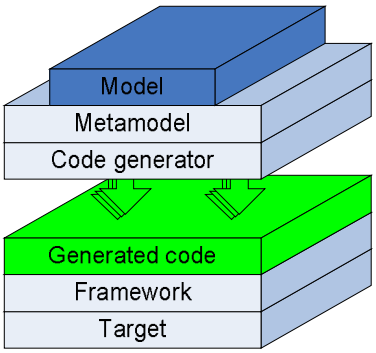

Figure 5. The basic architecture of DSM.

The primary driver for the NavApp framework is in the usability and extensibility of the framework. The interface for the framework is developed in such way that generating source code for the applications is straightforward. The framework encapsulates functions in a way that the generated applications mainly consist of function calls to the framework, in addition to utilizing the PyS60 framework. These requirements are materialised as a façade that hides all unnecessary details. Figure 6 represents all the relevant parts of the NavApp framework architecture.

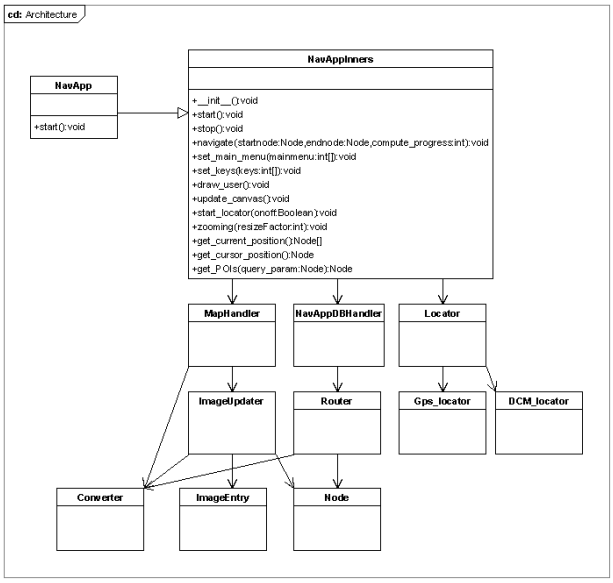

Figure 6. NavApp framework architecture.

The NavApp framework architecture is divided into four fundamental components:

- NavAppInners functions as a façade for the rest of the framework. It is the only NavApp-specific class that is accessed from the generated code.
- MapHandler is responsible for creating and handling the map, showing a route when navigating, and the other relevant visualization actions related to the map. In addition, MapHandler provides a set of callback functions to be used by the other components for displaying info on the map. Currently, pre-rendered images provided by the OSM sub-project, Tiles@home [9], are utilized as a map instead of rendering the map from the OSM data.
- NavAppDBHandler is responsible for handling the navigation data, computing route and also conducting queries directly to the OSM data for POIs etc. In addition, NavAppDBHandler provides facilities for storing additional data to the database.
- Locator is responsible for positioning. Positioning is performed by utilizing two of the most widely used positioning technologies, GPS and WLAN. WLAN positioning is based on an inbuilt DCM [6].

3.2 Architecture of the NavApp Applications

The code generator for the DSML for S60 [1, pp. 160-185] generates applications running on a state machine. Each entity is generated as a state realized by a function, where the states maintain a reference to the next state. In order to incorporate code generated from the DSML for NavApp, the generated code has to conform to that state machine. Therefore, the entity that initializes NavApp is generated similarly as any other state in the state machine generated from the DSML for S60. Thus, the introduction of the NavApp entity into the existing code generator and the language does not have any particular impact on the other entities.

The actual implementation entity of the NavApp which the initialization function calls is generated as a class that inherits the NavAppInners (see Figure 6). The NavApp maintains its own internal state. Similar to the code generated from DSML for S60 models, NavApp internal states are generated as states that are realized as functions, where all the functions maintain a reference to the next state.

Sub state machine and keyboard and menu definitions are generated differently. Whilst the sub state machine is generated as states running on the state machine, keyboard and menu definitions are generated as an encapsulated state in the sub state machine.

4. Discussion

The current version of the language is still immature and not as polished as possible, thus it requires one to become familiar with it. Therefore, the consideration of end-user driven development is still a matter of debate.

Currently, the language and framework are in active development. We are adding, among others, a possibility of utilizing an external positioning server in order to enable multi-person positioning and to enable the development of multi-user position-based games and to bring a social media dimension to the navigation applications.

5. Conclusions

In this paper, the work in progress in the development of the DSML for NavApp and its supporting framework is presented. The language is developed in a way that could enable non-programmer end-users to actively participate in the development of navigation applications or to develop applications completely by themselves. By utilizing the presented language, end-users can instantly experience the impact of changes in the model as the provided code generator enables complete code generation from the models.

[1] Kelly, S. and Tolvanen, J-P, “Domain-Specific Modeling: Enabling full code generation”, John Wiley & Sons, ISBN 978-0-0470-03666, 2008, 427 p.

[2] J. Bosch, “Design & Use of Software Architectures: Adopting and Evolving a Product-Line Approach”, Addison-Wesley, 2000.

[3] S60.com, URL: http://www.s60.com/life [Visited at 9.7.2008]

[4] Nokia.com, Nokia N95, URL: http://www.nseries.com/products/n95/#l=products,n95 [Visited at 9.7.2008]

[5] OpenSreetMap, The Free Wiki World Map, URL: http://www.openstreetmap.org/ [Visited at 9.7.2008]

[6] Kemppi, P. Nousiainen, S, “Database Correlation Method for GSM Location”, Vehicular Technology Conference, VTC 2000, IEEE VTS 53rd, 2001.

[7] Metaedit+ WorkBench, URL: http://www.metacase.com/mwb/ [Visited at 9.7.2008]

[8] Nokia Research Centre, Python for S60, URL: http://opensource.nokia.com/projects/pythonfors60/ [Visited at 9.7.2008].

[9] Tiles@home, URL: http://tah.openstreetmap.org/ [Visited at 9.7.2008]