Modelling Feature Interactions in Mobile Phones

Louise Lorentsen
University of Aarhus and Nokia Research Center (visiting),
ext-louise.lorentsen@nokia.com

Antti-Pekka Tuovinen, Jianli Xu
Nokia Research Center, P.O.Box 407, FIN-00045 NOKIA GROUP, Finland
antti-pekka.tuovinen@nokia.com, jianli.xu@nokia.com

Abstract

A modern mobile phone supports many features: voice and data calls, text messaging, phonebook, calendar, WAP browsing, games, etc. All these features are packaged into a handset with a small display and a special purpose keypad. The limited user interface and the intertwining of logically separate features cause problems in the development of the UI software for mobile phones. In this paper, we look at the problem of feature interaction in the UI of Nokia’s handsets. We present a categorization of feature interactions and describe our approach to modelling feature interaction patterns that uses explicit behavioural models of features and interactive graphical simulation. We use Coloured Petri Nets as the modelling language.

1 Introduction

The context of this work is the development of the UI software for Nokia’s mobile phones. In this domain, the term feature means functionality of the phone that is accessible or visible to the user via the UI and implemented by software. The features are implemented by UI applications in the proprietary mobile phone UI software system. Feature interaction means a dependency or interplay of features. The interactions can be conceptually simple usage dependencies or more complex combinations of independent behaviours.

The development of the user interface software for mobile phones is a concurrent and highly distributed engineering process. There is also strong pressure for reusing SW components in as many products as possible. In this kind of environment, it is important to identify and clearly specify the right interactions between the separate features of the mobile phone at an early stage of the development. This helps to avoid costly delays in the integration phase of a set of independently developed features. Precise descriptions of the interactions are also needed when planning the testing of the UI software. The number and type of interactions that a feature has with other features are also indicators of the cost of developing the feature.

Currently, feature interactions are not systematically documented. Often the most complex interactions are not fully understood before the features are first implemented. The goals of this work are to identify categories of interactions that are specific to the domain and to create behavioural models that capture the typical feature interaction patterns in each category.
The heart of our approach is an executable behavioural model of the underlying UI architecture and the individual features. As the modelling language we use Coloured Petri Nets, a visual, both action and state-based specification formalism that is suitable for modelling concurrent activities and flows in complex systems [3,4,5]. They have precise executable semantics which makes it possible to simulate the behaviour specified by CPN models. The tool that we use makes it possible to add domain specific graphics for visualisation and interaction purposes.

2 Classification of Feature Interactions

Each mobile phone product follows a certain UI style that captures the UI design of a product family. It describes the physical structure of the UI and the basic mechanisms of user interaction and it has a relatively long lifetime. The UI specification of a product defines the features of the product by showing the UI design and by describing the detailed user interaction for each feature.

Feature interactions come from different sources. Category I of interactions is the use interaction between features. For instance, the task-oriented nature of the user interface requires that when browsing the phone numbers stored in the phone, a call can be made to a number directly from the browser. This represents an interaction between the ‘phonebook’ and ‘mobile originated call’ features that is necessary to deliver a smooth and seamless service to the user. When compared with PC software, the applications in the phone SW have much more these kind of hard-wired dependencies.

Category II comes from the need to share the limited UI resources (screen, keypad) between many features that can be activated independent of each other. Because of the prioritization of the users tasks (and the associated features), important events may interrupt less important activities. For example:

- an incoming call screens phonebook browsing for the duration of the call but the browser application does not know it,
- hang-up key stops search from phonebook (the browser is killed), and
- an incoming call suspends a game but the game is saved and it can be continued.

Category III involves interactions where one feature affects other features by making them unavailable or by modifying their behaviour in some other way. For instance, the ‘any key answer’ feature makes it possible to answer an incoming call by pressing any key on the keypad and the ‘key guard’ feature locks the keypad for accidental key presses. The combined effect of these features is that if ‘any key answer’ is enabled and ‘key guard’ is on, an incoming call can be answered only by pressing the ‘send’ (off-hook) key. However, once the call is open, ‘key guard’ is disabled for the duration of the call and then enabled again automatically.

The use interactions are thoroughly specified in the UI specifications and they are not problematic from the implementation point of view. However, the interactions of the categories II and III are much more difficult to manage in software design and implementation; they also lack systematic documentation. Therefore, we concentrate on modelling and documenting the typical feature interaction patterns that belong to the latter two categories.
3 Our Modelling Approach

Coloured Petri Nets (CP-nets or CPN) [3] is a graphical modelling language with a well-defined semantics allowing simulation of the behaviour specified by CPN models as well as formal analysis. In contrast to many other modelling languages, CP-nets are both state and action oriented. CP-nets has proven powerful for modelling of concurrent systems and a number of successful projects have demonstrated its usefulness in modelling and analysis of complex systems, e.g., [1,4,6,8]. We use the tool Design/CPN [7] that supports editing, simulation and validation/verification of CP-nets.

Figure 1 gives an overview of the CPN model by showing how it has been hierarchically structured into 14 modules (also referred to as subnets or pages). Each node in Fig.1 represents a subnet of the CPN model. An arc between two nodes indicates that the source node contains a substitution transition whose behaviour is described in the subnet represented by the destination node.

Fig.1. The hierarchy page

The CPN model consists of four main parts corresponding to four concepts of the phone UI software system: applications, servers, UI controller, and communication kernel. Applications implement the features. The CPN model presented here includes four features: ‘game’, ‘basic call’, ‘alarm’, and ‘key guard’ features. Servers implement the basic capabilities of the phone. Applications implement the behaviour of features by using the services of servers. The CPN model presented here includes two servers: ‘call’ and ‘power’ servers. Applications make the feature available to the user via a user interface. Servers which provide the basic capabilities of the applications do not have user interfaces. Servers and applications are communicating by means of asynchronous message passing. The messages are sent through the communication kernel.

The subnet Top depicted in Fig.2 is the top-most page of the CPN model and provides the most abstract view of the CPN model. The page consists of four substitution transitions corresponding to the four parts mentioned above. The detailed behaviour of UIController, Servers, CommunicationKernel, and Applications is modelled on subnets associated with the substitution transitions.

A CP-net is created as a graphical drawing with some textual inscriptions. In contrast to many other modelling languages CP-nets are both state and action oriented. A state of a CP-net is represented by means of places which are drawn as ellipses with a name positioned inside. The places contain tokens, which carry data values, in CPN
terminology referred to as *colours*. Each place has a type, in CPN terminology referred to as a *colour set* which determines the kind of tokens which can reside on the place.

Actions of CP-nets are represented as *transitions* which are drawn as rectangles with a name positioned inside. The transitions and places are connected by *arcs*. Transitions remove tokens from places connected by incoming arcs and add tokens to the places connected by outgoing arcs. The tokens removed and added are determined by *arc expressions* which are textual inscriptions positioned next to the arcs. In the Design/CPN tool, the inscription language is Standard ML.

The use of substitution transitions allow the user to relate a transition to a more complex CP-net. The idea is analogous to the module concepts found in many programming languages. Furthermore, CP-nets has the concept of *fusion places* which allow the user to specify that a set of places are identical even though they are drawn as individual places (possibly belonging to different subnets). Using these two constructs together with Design/CPN's ability to save and load subnets of a CPN model we have constructed a CPN model of the phone UI software system where features can easily be added and removed. Hence, large parts of the CPN model can be reused when we later model other products with new features.

In addition to a graphical representation, CP-nets have formally defined semantics which makes it possible to simulate the behaviour specified by the CPN model. Design/CPN provides facilities for automatic simulations as well as interactive (step-by-step) simulations. However, the models we create are simulated and discussed in a forum of UI designers and software developers who are not experts in (or even familiar with) CP-nets. An important aspect of our work is therefore to extend the created CPN models with a layer of domain specific graphics which makes it possible to plan and control simulations and get feedback and information from these simulations without interacting directly with the CP-nets.
We have made two extensions to the CPN model that allow the visualisation of the current state of the CPN model and the behaviour of the CPN model during simulation. Firstly, the state of the phone as the user observes it on the handset is visualised via an animation of the display. Figure 4a shows a snapshot of the animation taken during a simulation of the CPN model. Secondly, the CPN model is extended with Message Sequence Charts (MSCs) [2] to be automatically constructed as graphical feedback from simulations. We chose to use MSCs in the visualisation because the SW designers already use them.

Fig. 4b shows an example of such a MSC automatically generated from a simulation of the CPN model. The MSC contains a vertical line for each of the applications and servers in the phone UI software system. The arrows between the vertical lines correspond to messages sent in the system. The communication sequence corresponds to a scenario where the mobile phone receives an incoming call while the user is playing a game (an interaction between the ‘game’ and the ‘call’ features). The sequence of events in the scenario is:

- The user selects a game from the menu (line 1)
- The game feature is notified and it requests the display (lines 2-3)
- The user selects a new game (line 4)
- The ‘game’ feature is notified and it changes the contents of the display accordingly (lines 5-6)
- An incoming call arrives. The ‘call’ server notifies the ‘call’ feature (lines 7-8)
- The ‘call’ feature requests the display (line 9)
- The display is currently in use of the ‘game’ feature. The UI controller interrupts the ‘game’ feature and after the interruption has been acknowledged the display is granted to the ‘call’ feature (lines 10-12)
The user rejects the call (line 13)
The ‘call’ feature is notified and the display is removed (lines 14-15)
The ‘game’ feature is resumed (lines 16-17)

Note that in the above scenario the UI controller is responsible for handling the interrupt (lines 10-12) and resume (lines 16-17) of features. The features do not have to know which features they potentially interrupt or are interrupted by. This makes it very easy to add and remove features from the CPN model without changing the subnets modelling the other features.

We have made two extensions to the CPN model to make it possible to control the simulations without directly interacting with the CP-nets. The first extension makes it possible to control simulations by clicking the keys of the image of the mobile phone in Fig. 4a. The second extension makes it possible to set up a scenario to be simulated. The scenario is specified as an ordered series of events. Figure 5 shows how the scenario corresponding to the MSC in Fig. 4b (where the mobile phone receives an incoming call while the user is playing a game) is specified. In this way it is possible to inspect interesting scenarios without manually pressing the keys of the mobile phone in Fig. 4a.

We have made two extensions to the CPN model to make it possible to control the simulations without directly interacting with the CP-nets. The first extension makes it possible to control simulations by clicking the keys of the image of the mobile phone in Fig. 4a. The second extension makes it possible to set up a scenario to be simulated. The scenario is specified as an ordered series of events. Figure 5 shows how the scenario corresponding to the MSC in Fig. 4b (where the mobile phone receives an incoming call while the user is playing a game) is specified. In this way it is possible to inspect interesting scenarios without manually pressing the keys of the mobile phone in Fig. 4a.

---

**4 Summary and Further Work**

The current model provides the basic UI infrastructure where we can plug in features. The model provides interactive and automatic graphical simulation. We have already identified some basic interaction patterns in the ‘classic’ UI style of the 6210 phone.

We are now adding more features to the model to build a comprehensive set of interaction patterns. One important task will be to link the interaction patterns to existing implementation patterns. Possible uses of the models include giving ideas to the UI architecture development work and regression testing when changing the logic of the features included in the model.

**References**


