Using Meta-modelling for Construction of an End-User Development Framework

Erlend Stav[1], Jacqueline Floch[1], Mohammad Ullah Khan[2], and Rune Sætre[2]

[1] SINTEF ICT, NO-7465 Trondheim, Norway
{Erlend.Stav,Jacqueline.Floch}@sintef.no
[2] NTNU, NO-7491 Trondheim, Norway
mukhan@item.ntnu.no, satre@idi.ntnu.no

Abstract. A main activity in meta-design is the creation of design spaces allowing problem owners to act as system developers. Meta-design is a conceptual framework; it does not provide concrete design space solutions or engineering guidelines for constructing tools that support design spaces. This paper discusses the applicability of a model-driven engineering approach for the realization of an end-user service composition framework, in line with the conceptual meta-design framework. We report our experience of using meta-modelling techniques as supported by the Eclipse Modelling Framework (EMF) family of tools. In our work we found that meta-models are well-suited to formalize the composition language, and the core parts of the EMF framework are useful to represent the language elements and user-made compositions both at design and runtime. Although EMF-based tools exist for creating visual editors, we found that in our case these did not map well to the visual notation we selected for our end-users.

Keywords: End-User Development, Meta-Design, Meta-Modelling, Model driven Engineering, Eclipse Modeling Framework.

1 Introduction

Our research starts with the vision of mobile pervasive computing, i.e. environments where objects are becoming increasingly intelligent and provide information and services to the user when and where needed. Tailoring the user environment to exactly what the user wants is challenging and requires a good understanding of individual needs. While several ambient intelligence approaches combine gathering of context and user activities with reasoning techniques to adapt environments to users, the vision where computers act as intelligent assistant "agents" is still an unrealistic promise [1] . We propose instead to empower the users so that they themselves can develop or adapt applications to their own needs and tasks in mobile pervasive environments. More specifically, we seek to develop a framework for end-user service composition. We see two main reasons for selecting services as a basis for end-user development. One is technical: the principles of the Service Oriented Architecture (SOA) are widely applied in the construction of software systems in mobile pervasive environments. SOA supports the dynamic composition of systems from loosely coupled functional entities (specified as services), which fits the needs of pervasive computing where resources can be represented as discoverable services and dynamically added to a system as they appear. The other reason relates to user understanding. SOA provides a paradigm shift in the way we think of software systems. Services are decoupled from the system realization and rather represent activities or results (i.e. a kind of consumables). Thus services are close to a human way of thinking in the real world, and SOA has the potential to reduce the gap between idea and system construction also for people without IT expertise.

In order to provide support for end-user composition, we adopt a meta-design approach [2]. *Meta-design is a conceptual framework that extends the traditional notion of system development to include users as co-designers, not only at design time, but throughout the entire life-cycle of the system* [3]. An important concern is that the user needs are not static. The users learn while using a system, and their needs evolve. It is therefore important to involve the users not only during system design, but also after system deployment. Meta-design describes an ecosystem for the collaboration between developers and users, with the *seeding, evolutionary growth and reseeding* (SER) process model as a central element. Seeds are initial system entities designed through participatory design activities involving developers and users. Seeds can grow, i.e. evolve, following the tailoring of the system by users. Finally, reseeding is about the enhancement of the initial system to integrate changes. The concepts of meta-design fit well in the context of end-user service composition: services map to the concept of seeds, user extensions through service composition map to evolutionary growth, and finally the creation of new services based on user compositions and new needs emerging during composition map to re-seeding.

A main activity in meta-design is the creation of a design space that supports the ecosystem for collaborative design. Meta-design is a conceptual framework. It does not provide concrete design space solutions or engineering guidelines for constructing tools that support design spaces. This paper discusses the applicability of a model-based approach for the realisation of an engineering framework for end-user service composition, in line with the conceptual meta-design framework. We describe our approach to the development of the model-based framework UbiSys and illustrate the usage of UbiSys for the end-user extension of a case application. Finally we discuss our experience of using meta-modelling techniques as well as the EMF technology.

2 Related Work

Various technical approaches have been exploited for the creation of End-User Development (EUD) frameworks [4]. Task specific Programming Languages, TSPLs, were advocated by Nardi [1] for two main reasons: 1) The concepts of the language relate to the task domain and thus are easy to understand by users familiar with the domain; 2) The language supports high-level operations related to the task domain and thus the user can express the desired system functionality without using low-level operations. Nardi also points out two main drawbacks of TSPLs: 1) It is expensive to build different TSPLs for different needs and computer usages; 2) The definition of multiple languages may require the users to learn different interfaces. In our work, we define a Domain-Specific Language (DSL) to support the development of applications adapted to the user tasks in mobile pervasive environments. The proposed DSL is therefore close to a TPSL. We may benefit from the advantages of using TSPLs, but have to face similar drawbacks. The complexity of realising a DSL-based end-user framework is one of the main issues in our work: we investigate the application of model-driven software engineering approaches and technologies for that purpose. Related to the second drawback, we differentiate between the composition concepts common across several domains, e.g. sequential service execution, and the service concepts of the application domain. In other words, we provide a single composition interface to the end users. The variation lies in the service abstractions (we call them building blocks) that need to be parameterized during composition.

Several approaches can be applied to develop a DSL [5]. We exploit existing techniques. A first issue is the identification of the domain concepts. Our work has addressed three application domains: city exploration [6], mobile telecom services [7] and mobile asset management. The separation between the composition concepts and the application domain services is a bit similar to that introduced in AgentSheets [8]. Although the agent-based approach of AgentSheets differs from ours, it also supports two programming levels: a domain-oriented language for defining the behaviour of agents, and domain-oriented agents to be used in domain-construction kits. The former corresponds to our composition, the latter to the building blocks.

A second issue in the development of a DSL is the design of the language itself. Since the composition model is to be transformed to an executable program that orchestrates the composed services, precise language semantics are important. Precise semantics are also needed for the construction of advanced end-user engineering tools, such as simulation and validation tools. We exploit meta-modelling that was found to be a good tool for the specification of DSLs in terms of expressive power, flexibility, constraints and clarity of the semantics [9]. We thereby avoid building a notation upon any existing software engineering modelling language, e.g. UML, because their focus on software professionals is likely to not suit non-IT experts.

Finally, a third issue is the construction of tools. To that end, we explore model-driven engineering (MDE) frameworks. MDE is an approach to software development where models are given a central role in the development process, and where the models are used directly to derive implementation artefacts [10]. Meta-modelling is usually used to define the modelling language in MDE approaches. Transformations, both model-to-model and model-to-text, are used to generate implementation artefacts. Recently, using models directly at runtime has also received some attention from the research community [11]. While MDE has principally been used in a professional software engineering context, [12,13] are examples of work closer to our own, where MDE is applied for to the creation of an end-user development framework.

Also related to our work, a number of end-user frameworks have newly been launched empowering mobile users to develop mobile applications themselves, for instance Google's App Inventor framework[1], Microsoft's TouchDevelop[2], NFC Task Launcher[3] or atooma[4]. As mobile devices are becoming more powerful in terms of computing and memory resources, and touchscreen technologies facilitate the construction of user-friendly interfaces, we expect that mobile software development will also get more accessible for all. Similar to the end-user frameworks for desktop environments, the mobile frameworks currently proposed adopt different language abstraction levels, i.e. programming vs. composition, and different development platforms, i.e. desktop vs. mobile tools. None of them explicitly support the extension of the framework by domain developers (i.e. the re-seeding step in the SER model).

3 Research Approach

Our research follows the design science paradigm [14]. While behavioural-science approaches focus mainly on the use and benefits of a system implemented in an organization, design science approaches develop and evaluate IT artefacts intended to solve identified organizational problems. Developing such artefacts requires domain knowledge and justification in form of proper evaluations. Design-science suggests an iterative work process allowing a gradual understanding of the problem to be solved and improvement of the solutions. It does not impose any concrete research and evaluation method since choice of method depends on the nature of the research problem and the type of the artefact being created.

The first step in our work was the specification of a set of scenarios that illustrate the concept of end-user service composition in mobile ubiquitous environments, and their evaluation and improvement through focus groups. The scenarios were used to: 1) elaborate the idea of end-user service composition and understand how it is perceived by the users; 2) identify an initial set of functionalities that users wish to create and a set of reusable services needed to create these functionalities. The scenarios were developed for three application domains related to the business areas of the research partners (see Section 2).

Following the specification of scenarios, our work has investigated alternative end-user notations. The notation for UbiComposer was selected to support both mobile-based and web-based scenarios. After the initial testing of a more complex notation through paper-prototyping, we decided to use a simple trigger-action sequence notation for the composition and a form-based presentation for the parameterisation of the services in a composition (see Section 5.4 for more

[1] The Site for Learning and Teaching App Inventor: `http://www.appinventor.org`
[2] TouchDevelop (Microsoft Research): `https://www.touchdevelop.com`
[3] NFC Task Launcher available on GooglePlay: `https://play.google.com`
[4] atooma: `http://atooma.com`

details). While the concepts of the proposed notation are inspired from the underlying concepts of visual flow languages that have proven to be successful in a number of end-users development environments, e.g. Lego MindStorm[5], the form-based approach is widely used for the parameterisation of online services and mobile applications. We have avoided a pure visual notation since it does not fit the pocket-size screens of mobile environments. As we will discuss later in this paper, the proposed UbiSys framework supports the realization of different end-user editors, and thus different notations may be provided in the future.

The focus of this paper, though, is on the development of engineering tools for end-user service composition. The main research problem is to find out what tools and technologies are well suited to building service composition environments for end users. This paper addresses the following questions:

1. How applicable is meta-modelling in the design of an engineering framework for end-user service composition?
2. Is it feasible to realize composition environments with existing model-driven engineering technologies, as exemplified by the Eclipse Modelling Framework (EMF) family of tools?
3. What are the architectural implications of meta-modelling and model-driven engineering technologies?

To answer these questions, we have prototyped and applied the service composition environment UbiSys. This paper discusses the experience we gained.

4 Overall Architecture

Figure 1 gives an overview of the UbiSys architecture, with the stakeholders in end-user service composition that we have identified, and with the tools and artefacts they use and create. We distinguish between two roles for meta-designers:

1. The *environment developers* create the service composition framework and the runtime environment for a specific composition approach, e.g. UbiSys. They are meta-designers that create tools for the composition design space.
2. The *domain developers* create reusable software services adapted to the needs of a particular domain, e.g. by adaptation of generic solutions. The services are created to fit the service composition framework. For instance, generic calendar services may be adapted to the needs of elderly people and to UbiSys. The domain developers are meta-designers that create seeds for composition in the design space, either as part of seeding or re-seeding steps.

These roles are motivated by the fact that creating tools for a design space requires different skills from creating services for composition by end-users. According to this separation, the developers in our own research activities were also organized in two teams: one on UbiSys and one on the City Explorer application example (see Section 6). In that way, we were able to identify initial difficulties that domain developers may face when taking the tools in use. Beyond developers, we also define two roles for the users:

[5] LEGO MINDSTORMS: `http://mindstorms.lego.com`

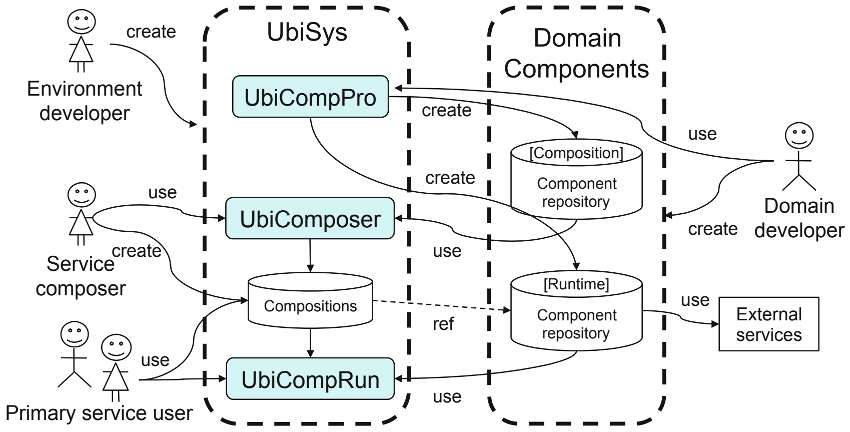


Fig. 1. System model

1. The *service composers* compose and tailor services into applications for service users. They test the service composition and eventually deploy it (or part of it) to one or several devices or servers.
2. The *primary service users* install, configure and use the services and applications created by service composers.

Similarly to [15], we consider end-user service composition to be an activity related to the development of software for personal use - unlike professional development targeting public use. It is, however, useful to differentiate between service composers and service users in several application domains. For instance, a caregiver may play the former role to create a service adapted to the needs of an assisted person, or a teacher to create a game for pupils. A user may play both the composer and user roles, e.g. a caregiver may participate in a service composed for assisted persons.

Our approach explores the application of a meta-modelling framework for the creation of a service composition framework by environment developers. The service composition framework itself, depicted as UbiSys in Figure 1, consists of three components:

1. *UbiCompPro* is a tool for domain developers allowing the creation of reusable components for composition. The domain developers implement components compliant with the runtime system in the framework, invoking domain services as needed. In addition they provide component descriptors that appear as building blocks for service composers to specify compositions from.
2. *UbiComposer* is a composition editor used by the service composers to select among the set of components defined using UbiCompPro and combine them into user-defined services and applications.

3. *UbiCompRun* is a runtime system for executing the services composed by the service composers. The runtime system interprets the composition models created using UbiComposer to control service execution, and invokes the right runtime components for the building blocks used in the composition.

The components developed using UbiCompPro correspond to the seeds of the meta-design framework. Both UbiComposer and UbiCompRun contribute to evolutionary growth by supporting the modelling and execution of user-created functionalities.

5 Framework Realization Using Meta-modelling

This chapter describes the meta-models defined by the UbiSys framework, and how these meta-models are used in the realization of UbiCompPro, UbiComposer and UbiCompRun. We chose to use the Eclipse Modeling Framework (EMF)[6] as the foundation for our realization because it is a mature open source framework with a whole family of tools built on top of it (e.g., it is the foundation of several commercial Eclipse based UML tools, including IBM's Rational Software Architect). In the first sub-section we give as background a short description of the EMF family of tools. We then describe the meta-models, before we give further details about our framework realisation based on these meta-models and EMF. The UbiSys framework and the City Explorer example application (See Section 6) are available as open source and documented on github[7].

5.1 EMF at a Glance

The Eclipse Modeling Project[8] is a top-level project in the Eclipse community that organizes the model-based development activities in the community. The foundation for most of tools that are sub-projects within Modeling is the Eclipse Modeling Framework (EMF). The core of EMF consists of three parts:

– eCore [16], the meta-meta-model of EMF, with supporting Java runtime libraries. The libraries contain APIs for managing model elements, and support for XMI-based persistence. EMF supports instantiation of meta-models based on generated Java classes, but also dynamic instantiation of non-generated classes using a generic, reflective API. This foundation provides interoperability between the tools based on EMF.
– EMF.Edit, a framework foundation for creating editors and views on top of the EMF models. This framework includes a command framework with a set of pre-defined commands that can be used to provide undoable operations on the model, like adding, deleting or moving model entities. Also, it provides facilities for defining the viewable structure and textual labels for model elements, giving a generic foundation for creating model views and editors.
– tools for the generation of runtime parts and a default model editor.

[6] Eclipse Modeling Framework (EMF): `http://www.eclipse.org/modeling/emf/`
[7] UbiSys and City Explorer source code: `https://github.com/UbiCompForAll`
[8] Eclipse Modeling Project: `http://www.eclipse.org/modeling/`

The runtime libraries and EMF.Edit were designed for use within the OSGi and Eclipse frameworks, but can also be used in stand-alone Java applications. From an EMF model, the generator tools of EMF enable the generation of:

– a Java representation of the model, including Java interfaces representing the model entity, implementation classes for these interfaces, and support classes including factories for creating instances.
– adapters based on EMF.Edit for presentation of the model elements.
– a fully functioning tree-based model editor that can be used from within Eclipse or in a stand-alone application using Eclipse Rich Client Platform.

While EMF can be considered to be a technology for developing abstract syntax, the Eclipse Modeling Project contains several tools for developing concrete syntaxes using EMF as their foundation. Among these are the Graphical Modeling Framework (GMF)[9] and Graphiti[10] for developing graphical modelling tools, and xText and EMF Text for developing tools using textual syntaxes.

5.2 The UbiSys Meta-models

The meta-models shown in this section were developed using EMF (using the standard EMF editors), but are conceptually independent of EMF and could be realized using other meta-modelling frameworks with a meta-meta-model similar to EMF's eCore. For the UbiSys tools, two meta-models were developed:

– The component descriptor meta-model (Figure 2) is used to model libraries of building block descriptors using the UbiCompPro tool. These libraries are used for providing the palette of building blocks in the UbiComposer tools.
– The user service meta-model (Figure 3) defines the abstract syntax for composing user services. The UbiComposer tool uses this to edit compositions, and the compositions are further used by UbiCompRun during runtime.

As shown in Figure 2, a descriptor library for components consists of a set of elements, which are building blocks, data types, or descriptors for domain objects (i.e. objects from the application domain). The main types of building blocks are triggers and steps, and each building block defines a set of properties. Elements and properties have user-friendly names that will be shown in UbiComposer.

The meta-model of the end-user's language is shown in Figure 3. It is used to represent the user services composed by the service composer. As shown in the figure, a user service is composed of one or more tasks, where each task has a trigger and a sequence of steps (actions). This corresponds to the trigger-action sequence used in the notations (see Section 5.4 for more details). Each building block (including trigger and step) has a number of property assignments, which can either be constant values, references to other properties, or references to domain objects. As shown in the figure, some concepts from the component descriptor meta-model are referenced to from this meta-model – e.g. each building block refers to its corresponding building block descriptor.

[9] Graphical Modeling Framework (GMF): `http://wiki.eclipse.org/GMF`
[10] Graphiti - a Graphical Tooling Infrastructure: `http://www.eclipse.org/graphiti/`

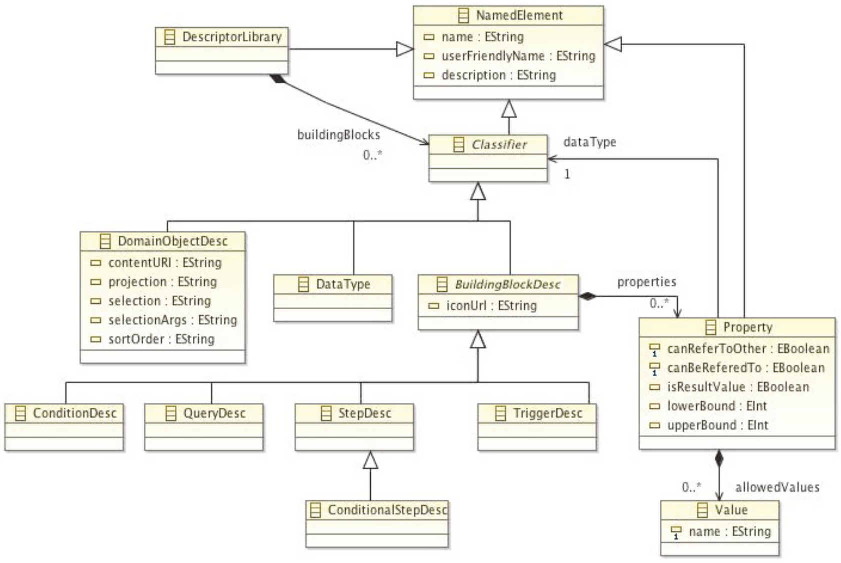


Fig. 2. Component descriptor meta-model

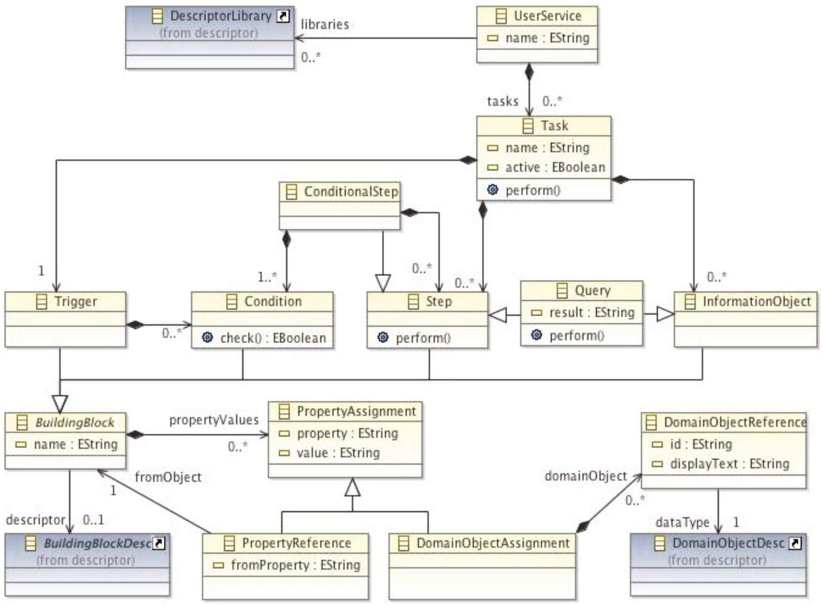
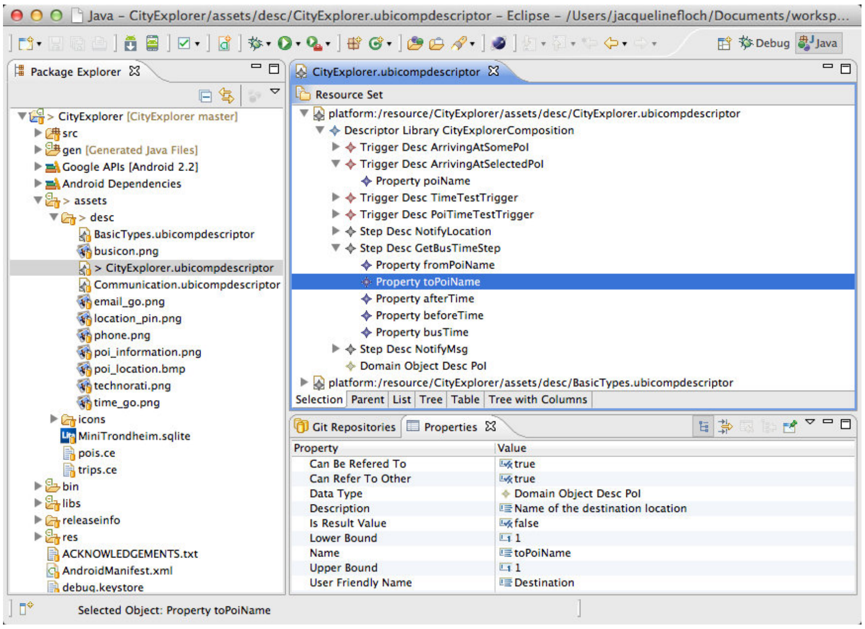


Fig. 3. User service meta-model


Java – CityExplorer/assets/desc/CityExplorer.ubicompdescriptor – Eclipse – /Users/jacquelinefloch/Documents/worksp...
Debug
Java
Package Explorer
CityExplorer.ubicompdescriptor
Selection
Parent
List
Tree
Table
Tree with Columns
Git Repositories
Properties

```
▼ > CityExplorer [CityExplorer master]
  ▶ src
  ▶ gen [Generated Java Files]
  ▶ Google APIs [Android 2.2]
  ▶ Android Dependencies
  ▼ > assets
    ▼ > desc
        BasicTypes.ubicompdescriptor
        busicon.png
        > CityExplorer.ubicompdescriptor
        Communication.ubicompdescriptor
        email_go.png
        location_pin.png
        phone.png
        poi_information.png
        poi_location.bmp
        technorati.png
        time_go.png
    ▶ icons
      MiniTrondheim.sqlite
      pois.ce
      trips.ce
  ▶ bin
  ▶ libs
  ▶ releaseinfo
  ▶ res
    ACKNOWLEDGEMENTS.txt
    AndroidManifest.xml
    debug.keystore
```

```
Resource Set
▼ platform:/resource/CityExplorer/assets/desc/CityExplorer.ubicompdescriptor
  ▼ Descriptor Library CityExplorerComposition
    ▶ Trigger Desc ArrivingAtSomePoI
    ▼ Trigger Desc ArrivingAtSelectedPoI
        Property poiName
    ▶ Trigger Desc TimeTestTrigger
    ▶ Trigger Desc PoiTimeTestTrigger
    ▶ Step Desc NotifyLocation
    ▼ Step Desc GetBusTimeStep
        Property fromPoiName
        Property toPoiName
        Property afterTime
        Property beforeTime
        Property busTime
    ▶ Step Desc NotifyMsg
      Domain Object Desc PoI
▶ platform:/resource/CityExplorer/assets/desc/BasicTypes.ubicompdescriptor
```

Property	Value
Can Be Refered To	true
Can Refer To Other	true
Data Type	Domain Object Desc PoI
Description	Name of the destination location
Is Result Value	false
Lower Bound	1
Name	toPoiName
Upper Bound	1
User Friendly Name	Destination

Selected Object: Property toPoiName

Fig. 4. UbiCompPro editor

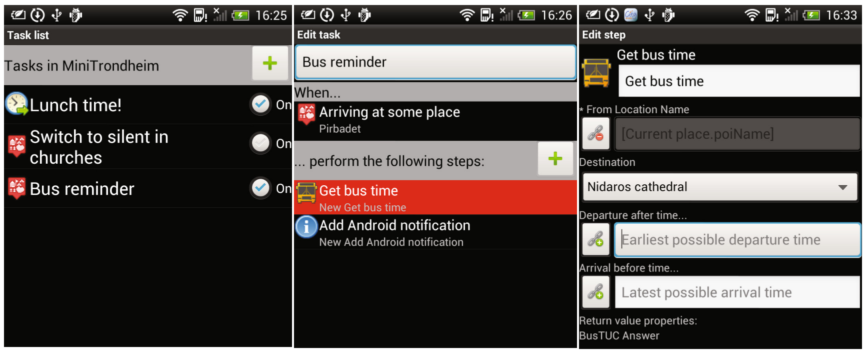
5.3 UbiCompPro Implementation

UbiCompPro (Figure 4) is fully generated from the component descriptor meta-model. The tool is an example of the standard tree-based editor generated by EMF. As the target users of UbiCompPro are software developers, our assumption is that the generated tree-based editor is suitable for the task of creating component descriptors. The descriptor files created using the tool are used directly in UbiComposer for displaying the palette of building blocks that are available to create compositions from.

Figure 4 shows a screenshot for the UbiCompPro editor where the developer can create entries for each building block in the library. The editor as shown in the figure is running in the Eclipse environment, with the current project expanded in the view at the left. In the figure, the descriptor library developed for the City Explorer example application case is selected and the property ”toPoiName” of the building block ”GetBusTimeStep” is being edited. The data type for that descriptor is a domain descriptor ”PoI” that supports access to data shared by the City Explorer application.

5.4 UbiComposer Implementation

UbiComposer is implemented in a mobile version for the Android platform (Figure 5) and in a web version (not depicted in this paper). At the top level of the composition, the service composer can define one or more tasks, where each


Task list

Tasks in MiniTrondheim

Lunch time! On

Switch to silent in churches On

Bus reminder On

Edit task

Bus reminder

When...

Arriving at some place
Pirbadet

... perform the following steps:

Get bus time
New Get bus time

Add Android notification
New Add Android notification

Edit step

Get bus time

Get bus time

* From Location Name

[Current place.poiName]

Destination

Nidaros cathedral

Departure after time...

Earliest possible departure time

Arrival before time...

Latest possible arrival time

Return value properties:
BusTUC Answer

Fig. 5. UbiComposer for Android

task represents a sequence of actions that will be performed (automatically) when a specific trigger occurs. The set of tasks are shown in a list (left-most screenshot of Figure 5). When editing the details of a task, its trigger and the actions can be selected from the set of available building blocks available in the tool (as provided by the domain developer), and the actions can be organized into the sequence in which they should occur. This is done in a task detail editor with pop-up menus for building block selection (middle screenshot of Figure 5). Form-based editing is used for setting the parameters of each building block. Parameter values can be typed in, selected among constants, or linked to properties of other building blocks or domain objects from applications. This is done in a detail editor for each building block. The right most screenshot of Figure 5 illustrates different cases: while ”Current place.poiName” is a reference to another building block, ”Nidaros cathedral” is a value from the City Explorer application (see Section 6). The editors perform some validation, e.g. actions with missing values for required parameters are highlighted in red in the Android version.

The Android implementation of UbiComposer is partially generated and uses the core EMF libraries without EMF.edit. The in-memory representation of the composition directly uses the Java classes generated from the EMF metamodel, and the default EMF persistence mechanism is used to load and save the compositions to a file-based storage. Also, UbiComposer uses the EMF-generated classes for the component descriptors to provide the palette of building blocks, and for setting up the detail editors for each component. The rest of the editor, including the user interface providing the concrete notation is hand-coded using standard Android libraries.

The implementation of the web version uses the Google Web Toolkit (GWT)[11]. A main criterion for the selection of GWT is the availability of end-user friendly widgets such as text boxes, selection boxes, forms, Google Maps, calendars etc. In addition to GWT, third party widgets from SmartGWT and the Google Map library were also integrated on the client side. GWT RPC was used for the communication with the server. Tools exist that generate tree-based editors

[11] Google Web Toolkit (GWT): `https://developers.google.com/web-toolkit/`

(like UbiCompPro) for GWT from EMF meta-models[12]. However, the generated editor is far from our selected end-user notation, and we found it difficult to adapt the generated code for our purpose. The current implementation of the web-based version of UbiComposer was therefore hand-coded based on the concepts of the meta-models. It can directly use component descriptor libraries created using UbiCompPro.

GMF was initially considered for implementing UbiComposer, but we found that its strength primarily lies in the development of notations such as UML class-diagrams that are different from the form-based notation we wanted for UbiComposer. Graphiti was not yet available at the time of our choice, and textual syntax tools such as XText were not an option for our notation.

5.5 UbiCompRun Implementation

Two runtime approaches are supported: interpretation of compositions and transformation to code. The former was realised on Android as UbiCompRun for Android. The latter was realised as a transformation (currently manual) to Drools rules since the Drools engine is used by one of our industry partners. This paper does not provide further details on the transformation to Drools because this is a proprietary solution of our industry partner.

Although the implementation of UbiCompRun on Android is mostly hand-coded, it also exploits the core EMF libraries. More specifically, it also uses the same Java classes generated from the user service meta-model as UbiComposer.

The hand-coded parts include the definition of the Java interfaces and abstract classes of the runtime framework that the domain developers use to implement their runtime components. Also, they include the classes performing the interpretation of the service compositions and the invocation calls to the runtime components. As part of the implementation of runtime components, the domain developer must also provide a simple map between component descriptors and component implementation classes.

6 Application Example: City Explorer

City Explorer is a mobile Android application that was developed in order to assess the UbiSys composition framework. City Explorer supports the management and sharing of contents for city exploration, e.g. places and itineraries, and the navigation to places. In addition, it supports the creation of new functionalities by the user. For example, the user may add tasks for sharing information through social media, getting bus information to a place defined by City Explorer, or setting up a lunch meeting place with friends. To support such creation, a number of components (and building blocks) that the user can compose together were defined. Respective to the meta-design framework, both City Explorer and the set of building blocks map to seeds. New building blocks, i.e. new seeds, may be

[12] EMF SDK for GWT: `http://wiki.eclipse.org/EMF/GWT`

gradually added depending on the emergence of new ideas and needs. Support for end-user extension of City Explorer is realised using UbiSys:

- UbiCompPro is used to create building blocks. In our experimentation, we have created event triggers, e.g. "at a specific time", "at a specific place" or "at any place in a specific itinerary", and steps, e.g. "send SMS", "add post on Facebook" or "get bus time".
- The UbiComposer Android library is integrated in the mobile application code. Thus, UbiComposer can be invoked from the application.
- The UbiCompRun Android library is also integrated in the mobile application code. Thus the composed services can be activated from the application.

An important requirement in the extension of City Explorer was the ability to access to application data both during composition and runtime:

- The service composer may wish to specify an extension for a particular entity or set of entities defined by City Explorer, e.g. "when arriving at a church, switch my phone to the silent mode." To do so, the service composer needs access to the place classifications defined by the application during service composition.
- The executing code extension may also need to access application data, e.g. "when arriving close to one of my favourite places, give me a notification and display information about that place." The executing code needs to retrieve the set of favourite places (and possibly listens to changes made to that set). It also needs to retrieve information about a place when getting close to it.

UbiSys introduces the concept of "domain descriptor" that supports the creation of building blocks that access application data. Access to data requires the application to expose its data in compliance to the rules defined by UbiSys. Currently, UbiSys supports the Android Content Provider mechanism. In that way, Android application developers do need to learn any new mechanism to expose application data.

7 Discussion

The discussion provided in this chapter is based on our own experiences in applying meta-modelling and EMF to the development of the UbiSys framework and example applications. While the team working on UbiSys have obviously had a personal interest in succeeding with development, we do not have any bias regarding the use of meta-modelling or the EMF family of tools.

Most of the attention of the MDE research community has been on simplifying development for different groups of software developers. In our work, we found that MDE is also useful for the realization of end-user development tools:

- Meta-models were found useful for discussing the realization of the composition language for the selected end-user notation. The development of meta-models requires a precise definition of notation concepts. In addition it is a tool for seeking simplification of the models.

- MDE simplifies the task of environments developers, i.e. the realisation of the end-user composition framework.
- When meta-models are used all the way from design of building blocks, via composition, to runtime, consistency is enforced. This contributes to a smooth transition between the activities of domain developers and the activities of service composers in the design space.

On the other hand, we found meta-modelling inadequate for the rapid exploration of alternative notation concepts. Instead, visual prototypes (paper and quick SW mock-ups) are in our experience better suited for discussing and agreeing on the end-user composition language because they also provide the concrete syntax elements of the language. The different application domains addressed in our research gradually raised new requirements on the structural concepts needed for service composition. For instance the telco case added a requirement for if-then-else structures (not needed by other cases), while the mobile asset management case added a requirement related to the use of conditions in association with triggers. We were able to extend the initial meta-models to support these concepts, but have completed implementation of them. We have no experience so far on supporting extensions that would require more complex adjustments of the meta-model.

The basic EMF technology selected in our work to realize the model-driven engineering approach was found to be suitable for the "invisible" parts of the composition and runtime tools (i.e. the parts not exposed to end users):

- EMF supports the instantiations of meta-models (i.e. the creation of models) and persistent storage. We use EMF libraries and generated Java classes to support the specification of compositions based on the composition notation, and to implement import and use of component descriptor libraries in UbiComposer.
- Based the specification of a meta-model, EMF supports the generation of a tree-base editor for the creation of models. UbiCompPro is such an editor. The domain developers can easily install UbiCompPro as a plug-in in their development environment and create descriptors for the building blocks descriptors the service composer will choose among during composition.
- The EMF cross-platform support worked between Eclipse-based / Desktop Java, and Android. In the case of Android some repackaging of the EMF libraries was required, and also the full potential of EMF was not used. Although there is also support for automatic generation of GWT projects from EMF meta-models, we found it too difficult to integrate this with the end user-friendly widgets that we needed, and thus EMF libraries were not used in the web-based UbiComposer.

When starting the development of UbiComposer, we also looked for tools based on EMF that could assist in developing the visual parts of our notation. The main candidate we found at that time was GMF, but it was not selected because it did not match well to our selected notation. Also, it does not support the Android and web-based platforms, and thus would only have been useful on the desktop.

The different industry partners involved in our work had different requirements on composition and runtime. Using a MDE approach, the proposed notation models remain platform-independent and we were able to integrate the tools with other applications and middleware. The adoption of a meta-modelling approach and the EMF technology has provided us the flexibility to fulfil various architectural needs:

- Composition editors were developed both in native code for Android and as a web-based solution. The former enables integration of the editor with any Android app. The latter enables access to the editor on any platform.
- The compositions were both interpreted at runtime on Android and transformed to Drools rules. In the former case, as the editor is also available on Android compositions can be modified at runtime.

8 Conclusion and Further Work

The EUD community have mostly focused on the end-user perspective of EUD, and not so much on technical realization of the required tools. This paper describes a MDE approach to the realisation of such tools. It positions the approach with respect to the meta-design framework and reports our experience, mostly positive, in adopting MDE and the EMF technology. Our work is a first step in the realisation of end-user tools. Relevant future areas of work include:

- *Validation support.* End-users lack knowledge in software engineering practices. Support for creating correct compositions and avoiding errors is therefore a critical concern. We intend to investigate how the EMF validation framework can be exploited to check the models.
- *Simulation support.* Most service compositions created for mobile pervasive computing do not occur at once, but are triggered in a specific context. Thus, differently from spread-sheet applications or EUD game environments, the end user cannot observe the effect of a composition at once. We intend to build simulation and debugging tools allowing end users to test the compositions and search for the causes of eventual errors.
- *Guidance to domain developers.* The proposed framework does not provide any guidelines for the specification of building blocks adapted to the level of expertise of non-IT experts. Few software developers are familiar with the discipline of end-user development. We intend to enhance the tools with guidelines based on earlier experience such as found in [17,18]
- *Adaptation to emerging technologies.* Another relevant work is the exploration of new tools to realize visual notations, such as the recent additions to the EMF family of tools, including Graphiti and Extended Editing Framework.
- *End-user evaluation.* The tools were improved based on feedback from project participants. A more extensive evaluation including both external developers and end-users is required.

Acknowledgement. Our research has been performed in the Norwegian Research Council (NFR) project UbiCompForAll in cooperation with the EU IST project SOCIETIES (contract 257493).

References

1. Nardi, B.A.: A small Matter of Programming. The MIT Press (1993) ISBN: 9780262140539
2. Fischer, G.: End-User Development and Meta-Design: Foundations for Cultures of Participation. Journal of Organizational and End User Computing 22(1), 52–82 (2010)
3. Fischer, G., et al.: Meta-design: a manifesto for end-user development. Communication of ACM 47(9), 33–37 (2004)
4. Sutcliffe, A., Mehandjiev, N.: Special issue on End-User Development. Communications of the ACM 47(9) (2004)
5. Mernik, M.: When and How to Develop Domain-Specific Languages. ACM Computing Surveys 37(4) (2005)
6. Floch, J.: A Framework for User-Tailored City Exploration. In: Costabile, M.F., Dittrich, Y., Fischer, G., Piccinno, A. (eds.) IS-EUD 2011. LNCS, vol. 6654, pp. 239–244. Springer, Heidelberg (2011)
7. Sanders, R.T., Mbaabu, F., Shiaa, M.M.: End-user Configuration of Telco Services. In: Proc. of 16th Int. Conf. Intelligence in Next Generation Networks: Realising the Power of the Network (ICIN 2012). IEEE (2012) (10.1109/ICIN.2012.6376036)
8. Repenning, A., Ioannidou, A.: Agent-based End User Development. Communications of the ACM 47(9) (1994)
9. Weisemöller, I., Schürr, A.: A Comparison of Standard Compliant Ways to Define Domain Specific Languages. In: Giese, H. (ed.) MoDELS 2007 Workshops. LNCS, vol. 5002, pp. 47–58. Springer, Heidelberg (2008)
10. Stahl, T., Völter, M.: Model-driven software development: technology, engineering, management. Wiley, Chichester (2006)
11. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. IEEE Computer 42(10) (2009)
12. De Silva, B., Ginige, A.: Meta-model to support end-user development of web based business information systems. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 248–253. Springer, Heidelberg (2007)
13. Fogli, D., Parasiliti Provenza, L.: End-user development of e-government services through meta-modeling. In: Costabile, M.F., Dittrich, Y., Fischer, G., Piccinno, A. (eds.) IS-EUD 2011. LNCS, vol. 6654, pp. 107–122. Springer, Heidelberg (2011)
14. Hevner, A.R., March, S.T., Jinsoo, P.: Design Science in Information Systems Research. MIS Quarterly 28, 75–105 (2004)
15. Ko, A.J., et al.: The State of the Art in End-User Software Engineering. ACM Computing Surveys 43(3) (2011)
16. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework, 2nd edn. Addison-Wesley Professional (2008)
17. Myers, B.A., Pane, J.F.: Natural Programming Languages and Environments. Communication of ACM 47(9), 47–52 (2004)
18. Repenning, A., Ioannidou, A.: What makes end-user development tick? 13 design guidelines. In: Lieberman, H., Paterno, F., Wulf, V. (eds.) End-User Development. Springer (2006) ISBN 1-4020-4220-5