2011-9

Self-adaptive application for indoor wayfinding for individuals with cognitive impairments

Basel Magableh
Technological University Dublin, 453543@tudublin.ie

Stephen Barrett
Trinity College, Stephen.Barrett@tcd.ie

Recommended Citation

Magableh, B., Barrett, S. (2018) Self-adaptive application for indoor wayfinding for individuals with cognitive impairments," in The 24th IEEE International Symposium on Computer-Based Medical Systems, ser. CBMS 2011, vol. 1, Lancaster, UK, June 2011, pp. 45–49. doi.org/10.21427/my07-et32

2011

Self-adaptive application for indoor wayfinding for individuals with cognitive impairments

Basel Magableh

Self-adaptive application for indoor wayfinding for individuals with cognitive impairments

Basel Magableh and Stephen Barrett
Distributed Systems Group, School of Computer Science and Statistics
Trinity College Dublin, Ireland
Emails: magablb@cs.tcd.ie, stephen.barrett@cs.tcd.ie

Abstract

This article focuses on describing a Model Driven Architecture (COCA-MDA) approach that facilitates the development of self-adaptive application for indoor wayfinding for individuals with cognitive impairments. COCA-MDA provides the following benefits: 1) It enables the architecture to anticipate several behavioural variations based on the context and the specific needs of the individuals with cognitive impairments. 2) It enables the application to proactively anticipate or reactively address unforeseen changes through support by a dynamic-decision making and policy framework. The policy framework is based on a stable description of software models and proprieties. 3) It can decompose the application into several architectural units to allow developers to decide which part of the architecture should be notified when a specific context condition occurs.

1 Introduction

Some of the challenges for individuals with cognitive impairments in wayfinding are remaining oriented, recalling routines, and travelling in unfamiliar areas while relying on limited cognitive capacity. Whereas people without disabilities often use maps or written directions, either as navigation tools or for remaining oriented, the cognitively impaired population is very sensitive to issues of abstraction (e.g. icons on maps or signage), which presents the application designer with the challenge of tailoring navigational information to each specific user and context. With the capacity to move and the desire to be socially included, mentally/cognitively disabled individuals who are independently mobile but have difficulties reaching their intended destination might benefit from the self-adaptive application proposed in this study. A self-adaptive application modifies its own structure and behaviour in response to changes in its operating environment [1]. Hirschfeld et al. [2] consider context to be any information that is computationally accessible and upon which behavioural variations depend.

Adaptation is caused by context changes. Whenever the system's context changes, the system has to decide whether to adapt. Context-dependent behaviour variation is a computational entity that provides information about the changes in its artefacts, the so called system behaviour parts [2]. The complexity of these behaviour variations lies on the fact that they can occur separately or in any combination. Therefore, a formal procedure is required to analyse them and separate their various concerns. Moreover, a formal procedure to model these variations is needed. These analysis and modelling procedures can reduce the complexity of context-aware application modelling. In this way, it facilitates the development process and componentisation of the system into several behavioural parts. These parts can be used dynamically to modify the application behaviour based on the execution context.

This article focuses on describing a Context Oriented Component-based Application Model Driven Architecture (COCA-MDA) approach for developing a self-adaptive application for indoor navigation (IWayFinder). COCA-MDA organises the architecture into two casually connected layers: the base layer, which provides the application's core structure, and the meta-layer, where the components-based model (COCA-components) are located. The COCA-component is a unit of behaviour that provides composable units of behaviour. Hereafter, these COCA-components are handled by the COCA-middleware [3], for application composition and adaption. The proposed self-adaptive application anticipates the context information that is delivered by the Cisco Mobility Services Engine infrastructure [4] that provides location-based, time-based, and profile-based contextual information.

The remainder of the article is structured as follows. Section 2 provides a comparative analysis of related studies. Section 3 demonstrates the COCA-MDA modelling process used by IWayFinder. The COCA-MDA phases are described in Section 4. Section 5 provides an overall description of the COCA-middleware. Section 6 evaluates IWayFinder in terms of energy use, context monitoring, and detection enhancement.

2 Related Work

An indoor wayfinding application was developed by [5]. The passive architecture integrates radio-frequency identification (RFID) tags for wayfinding. Once the tag is scanned, a request is sent via a GPRS connection to a back end server. The server processes the location data and sends the routing information on the form of a pre-stored images that describe several alternative directions on the path. For each decision point, the server provides the mobile device with four alternative images. The proposed architecture [5] has several drawbacks: 1) The passive RFID tags are hard for the cognitively-impaired individuals to locate. 2) Delivering the navigation directions in images increases the cognitive load for the user; in some cases the image on the screen is difficult to view. 3) The application does not consider the resource scarcity of the mobile device. 4) The application does not provide behavioural variations that consider unanticipated conditions. 5) The application is not able to manage the trade-off between resources cost and priority/importance to use them.

In the literature, there are several MDA approaches that target self-adaptive, context aware applications such as MUSIC [6], U-MUSIC [7], and Paspallis MDA [8]. The U-MUSIC methodology [7] adopts MDA to enable dynamic unanticipated adaptation based on a component model. U-MUSIC enables the developers to specify the application variability model, context elements, and data structure. Paspallis et al. [8] propose another MDA-based methodology that considers the context providers for the application: for each context provider, a plug-in or bundle is planned and designed during the design phase. The MUSIC development methodology [6] adopts a model-driven approach to constructing the application variability model. In MUSIC, middleware is used to resolve the variation points, which involves the election of a concrete component as a realization for the component type. A number of application variants can automatically be derived.

These approaches suffer from a number of drawbacks. First, it is well known that correct identification of the weight of each goal is a major difficulty for the utility functions. Second, the approach hides conflicts among multiple adaptation goals by combining them in a single, aggregate objective function, rather than exposing the conflicts and reasoning about them. On the other hand, it would be optimistic to assert that the process of code generation from models can become completely automatic and that the developer's role lies only in application design. Third, it is impossible for the developer to predict all possible variations of the application when unanticipated conditions will arise. In addition, mobile devices have limited resources for evaluating many application variations at runtime and can consume significant amounts of device resources. As result, the benefit gained from the adaptation is negated by the overhead required to achieve the adaptation. Fourth, mobile devices produce an architecture with a tight coupling between the context provider and the context consumer, which may cause the middleware to notify multiple components about multiple context changes. Finally, this form of the decision making process does not allow the user to control the adaptation effect.

CAMEL, proposed in [9] as a metamodel for adaptation in what was called "contextual adaptation", uses an insert or binding adaptation mechanism based on the JCOOL domain-specific language. CAMEL has no formal MDD methodology that has a generic life cycle that a developer could use. Regardless of these problems, JCOOL is better suited to the SMILE platform and the Java language. The A-MUSE methodology was introduced in [10] to integrate the MDA approach with a service-oriented architecture. The A-MUSE methodology shows novelty in its behavioural model transformation; however, no generic methodology was proposed. Behavioural anticipation and dynamic decision making are not supported by CAMEL.

In general, no approach or methodology has been proposed specifically for developing self-adaptive applications that consider the nature of context-dependent behaviour variation and anticipate context changes. Indeed, it is currently quite difficult to produce context-aware applications, especially in the presence of unforeseen changes in condition. Anticipating context changes using a model approach requires both a formal procedure to analyse and model those context changes and a dynamic-decision making process supported by middleware. In addition, the developers can design the system to proactively or reactively address unforeseen changes by providing a decision policy that triggers the adaptation whenever specific context values cross (lower or upper) boundaries. This process is easy to accomplish so long as the middleware is aware of which parts of the architecture (components) are affected by these context changes. Additionally, the predefined policy can provide the middleware with sufficient information to perform the adaptation.

3 Self-adaptive indoor wayfinding for individuals with cognitive impairments application

The Cisco mobility infrastructure has the ability to capture and employ contextual information about mobile assets. Contextual information can be collected automatically using the Wi-Fi connectivity of the asset (for example, laptops or Wi-Fi phones) or, for assets that do not have intrinsic wireless, by attaching radio frequency tags or QR-codes to the asset. QR-codes are a specific matrix barcode (or two-dimensional code) that is readable by dedicated QR barcode readers and camera phones [11]. The benefit of integrating the application with the Cisco engine is the integration of several assets that provide the contextual information. QR-codes or RFID tags are placed at the decision points (DPs) (such as hallway intersections, exits, doors, elevators, or entrances to stairways) identified by the Cisco engine.

A user enters the building and points the mobile phone's camera at any of the QR-codes available at the DP. After reading the encoded URL in the QR-codes, the Cisco engine then provides the required navigation information and instructs the user. To overcome the challenges of image rendering, the proposed self-adaptive application uses an augmented reality browser (ARB) to display the navigation directions. The browser displays the directions on the physical display of the tool's camera. Using the device camera, the system reduces the cognitive load and increases the user's ability to realize the desired route. In addition, the application is able to provide the user with time-based events such as the opening hours of the building, lunch time, closing hours of the offices, location access rights that control the entrance of users to certain locations, and any real time alarm events. Moreover, the infrastructure support allows several persons to monitor and collaborate with the user en route. Assuming the context information is delivered by the Cisco infrastructure, the following anticipation scenarios are proposed:

A1: Self-tuning The application must track the user's path inside the building. When a decision point (DP) is reached, the application places a marker for each DP the user passed. If the user is unable to locate the a decision point in the building, the application must be able to guide the user towards a safe exit. The route directions can be delivered to the user in several output formats: video, still image, and voice command. The application should change the direction output while also considering the device resources and the level of cognitive impairment of the individual.

A2: Self-recovering Assuming the user is trapped in a lift with no GPRS connection or in the case of a fire, the fire alarm is raised, the application is notified, and the application adopts the shortest path to the nearest fire exit. In both cases, the application submits the user's current coordinates and an emergency help message to the emergency number, parents, career team, and security staff. The communication is achieved using the available connection, regardless of the resource cost, to alert any nearby devices about the emergent need for help. If no connection is made, the device emits an alarm sound and increases the device volume to maximum. The security staff or fire-fighters receive the emergency message and can view the CCTV video to identify the floor on which the user is trapped. When the CCTV system locates the user, full information about the user is displayed, including a personal and health profile. At the same time, the application guides the user to a safe exit using a preloaded path (in case the CCTV camera is disabled and the services engine is off). Fire-fighters can use the received message to locate the user in the building.

4 Generic Phases of COCA-MDA

The COCA-MDA follows the principles of the object management group (OMG) model-driven architecture. In MDA, there are three different views in the software: the computation-independent view (CIV), the platform-independent view (PIV), and the platform-specific view (PSV). The CIV focuses on both the environment of and the requirements of the system and hides the details of the software structure and processing. The PIV focuses on the operation of the system and hides details that are dependent on the deployment platform. The PSV combines the CIV and PIV, with an additional focus on the details of the use of a specific platform by the software system [12]. COCA-MDA has adopted the component collaboration architecture (CCA) [13] at the PIV phase by partitioning the software into two views: the structure view and the behaviour view. The structure view focuses on the core components of the self-adaptive application and hides the context-driven components. The behaviour viewpoint focuses on modelling the context-driven behaviour of the component, which may be invoked in the application execution at runtime. The design of a context-aware application according to the COCA-MDA approach generally involves the six phases shown in Figure 1.

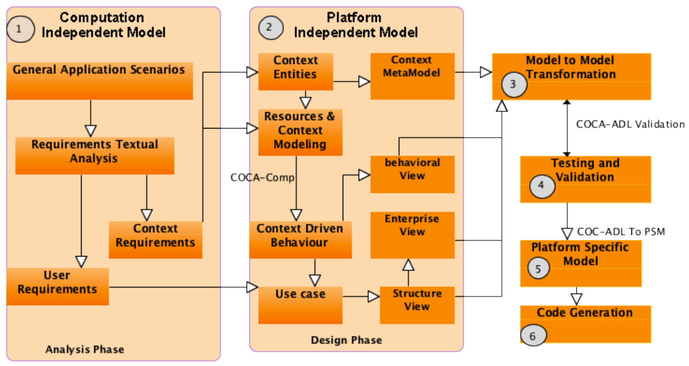

Figure 1. Context-oriented Component-based Application Model-Driven Architecture (COCA-MDA)

Analysis: The requirements of the system are modelled in a computation-independent model (CIM), thus describing the situation in which the system will be used. A CIM is a model of a system that shows the system in the environment in which it will operate, and thus, it helps to present exactly what the system is expected to do. It is useful not only as an aid to understanding a problem, but also as a mechanism for predicting the exact behaviour of a software system as a result of runtime changes. The first COCA-

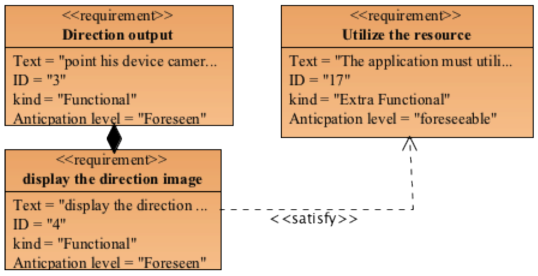
<<requirement>>
Direction output
Text = "point his device camer...
ID = "3"
kind = "Functional"
Anticpation level = "Foreseen"
<<requirement>>
Utilize the resource
Text = "The application must utili...
ID = "17"
kind = "Extra Functional"
Anticpation level = "foreseeable"
<<requirement>>
display the direction image
Text = "display the direction ...
ID = "4"
kind = "Functional"
Anticpation level = "Foreseen"
<<satisfy>>

Figure 2. Partial Requirements Diagram

MDA model is based on the requirements diagram in Figure 2. This diagram is used to classify requirements based on their type and anticipation level. The requirements are divided into functional and extra-functional. Hochmuller et al. refer to the technological and non-functional properties of software architecture as extra-functionalities [14]. Extra-functional extends the application behaviour in response to context changes. This work focuses on addressing extra-functionality as a collection of context-oriented components (COCA-components).

Modelling and design: Platform-independent model. The platform-independent view focuses on the operation of a system while hiding the details necessary for use of a particular platform. In this phase, the requirements diagram is combined into a use-case model. The use-cases describe the interactions between the software system and the actor. The system-dependent and environment-dependent behaviours are modelled as an extension of the functional use-cases. The functional use-cases are modelled in a class diagram describing the application core functions. The extended use-cases are modelled as another class diagram that describes the application's behavioural view. For example, the fire alarm use-case is a contextually-driven use-case that extends the application functionality to send an emergency message and to provide a route to the nearest fire exit.

The use-case diagram is split into two distinct class diagrams. The first diagram describes the basic application components that are executed regardless of the execution context. The core structure is integrated with the extra-functional class model in the final architecture model. The extra-functionality class diagram provides a detailed view of the application COCA-component and the COCA-middleware. In addition, these diagrams model the desired behaviour that can be used to anticipate context changes. Figure 3 shows a COCA-component modelled to anticipate the 'direction output'. The COCA-component implements a delegate objects and sub layers; each layer implements a specific context-dependent function. The COCA-middleware uses this delegate object to redirect the execution among the sub layers, based on the context condition.

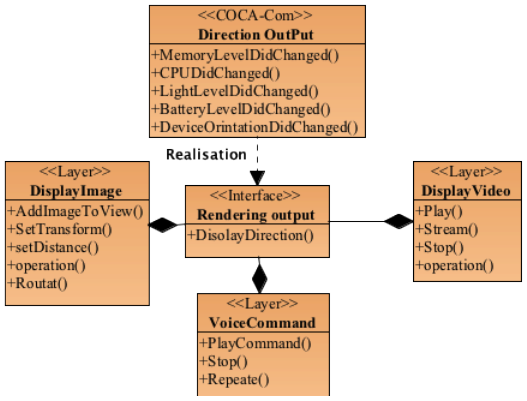

Figure 3. Direction Output Context Oriented Component

The application behavioural model is used to demonstrate the decision points in the execution that might be reached whenever internal or external variables are found. This decision point requires several parameter inputs to make the correct choice at this critical time. Using the activity diagram, the developers can extract numerous decision polices. Each policy must be modelled in a state diagram: textbfPolicy: Direction output This policy is attached to the 'direction output' COCA-component in Figure 3. The policy syntax can be described by the code shown in List 1.

Listing 1. Decision Policy 2

```
If ( direction is Provided && Available memory >= 50
&& CPU throughput <= 89 && light level >= 50
 && BatteryLevel >= 50) then {PlayVideo(); displayImage();
VoiceCommand();}
  Else If ( BatteryLevel < 50 || memory level < 50 || CPU >92)
  then {displayImage(); VoiceCommand();}
  else If( BatteryLevel < 20)
   then VoiceCommand();
```

The variant behaviour model is supported by a state-machine model that describes the application decision polices. The three models of the application are used as input for the next phase, model-to-model transformation.

Model-to-model transformation: The platform-independent model and behavioural model are translated into architecture description language (COCA-ADL). This phase includes model-to-model transformation and model verification for the application's structure and behaviour views. The COCA-ADL is implemented by extending the xADL schema, which is an extensible XML language. ArchStudio helps developers to model the architecture using three grouped models: activity diagram, state diagram, and structure diagram [15].

Testing and validating: Tests the model and verifies its fitness for the application goals and objectives.

Platform-specific model: The platform-specific model produced by the transformation is a model of the same system specified by the PIM; it also specifies how that system makes use of the chosen platform. A PSM may provide more or fewer details, depending on its purpose. A PSM will be an implementation if it provides all the information needed to construct a system and to put it into operation. Alternatively, it may act as a PIM used to further refine the PSM so that it can be directly implemented.

Code generation: Model-to-text includes model-to-text transformation deployment and execution verification. The COCA-ADL XMI code is transformed into the implementation language.

5 Overview of the COCA-middleware

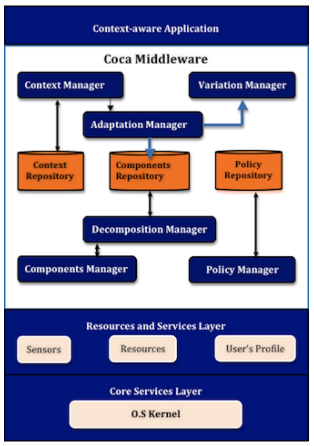

Figure 4. COCA-platform architecture.

The COCA-platform offers a context-aware middleware environment for adjusting the application's behaviour dynamically. Figure 4 shows the COCA-middleware architecture. The platform is layered into four major layers. Each layer provides an abstraction of the underlying technology. Each layer is totally platform independent of any given technology. The first layer represents the context-aware application. It provides the user with GUI, functional properties, and non-functional properties. The second layer in the platform represents the COCA-middleware. It has the middleware components. Fuller details of the individual components and subcomponents can be found in Figure 4. The OS sensor retrieves information about the OS. Function calls are used to retrieve information about CPU, memory, and disk space [3].

6 Experiment evaluation

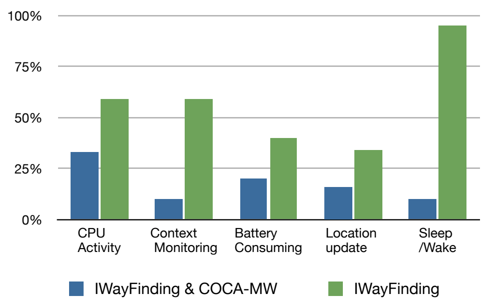

Figure 5. Energy usage for IWayFinder application.

The 'direction output' COCA-component registers itself with the context manager to be notified when the *BatteryLevelDidChanged, CPULevelDidChanged, MemoryLevelDidChanged, DeviceOrientationDidChanged and/or LightLevelDidChanged*. When the context manager notifies *[PostNotfication:BatteryLevelDidChanged]* to the 'direction output' component, the adaptation manager reads the attached decision policy in list 1. Based on the policy action, the adaptation manager calls the delegate object *DisplayDirection* which forwards the method invocation to the desired sub layer, based on the battery level. If the battery level $< 20\%$, then the adaptation manager activates the sub layer VoiceCommand to adapt this context condition.

The IWayFinder application has been implemented in two different versions, with and without the COCA-middleware. The battery life has been measured by running each version on an iPhone 4 device. The experiments show that the COCA IWayFinder application saved battery consumption by 13%, despite its self-adaptability, as shown in Figure 5. One of the expected benefits of using COCA-MDA in structuring the self-adaptive application is the enhancement of the context monitoring and detection processes. The IWayFinder implementation without the COCA-platform consumes more energy during context monitoring, thus draining the battery faster, because context changes are sent to a large subset of components. On the other hand, when the same application adopts the COCA-middleware, the application is able to adapt its behaviour and enhance both context monitoring and detection. The adaptation/configuration time and the context monitoring time are shown in Figure 6.

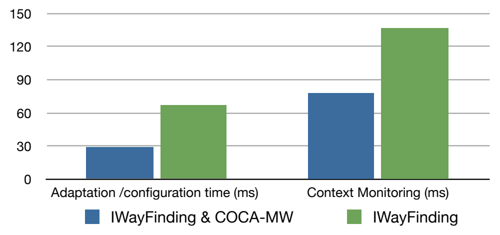

Figure 6. Adaptation time (ms)

7 Future Work

The COCA-MDA requires an improvement regarding support for both requirement reflection and modelling requirements as runtime entities. The requirement reflection mechanism requires support at the modelling level and at the architecture level. Reflection can be used to anticipate the evolution of both functional and non-functional requirements. The decision policies require more development regarding policy mismatch and resolution. This is in line with an improvement in terms of self-assurance and dynamic evaluation of the adaptation output.

References

[1] P. Oreizy, M. Gorlick, R. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf, "An architecture-based approach to self-adaptive software," *Intelligent Systems and Their Applications, IEEE*, vol. 14, no. 3, pp. 54–62, 1999.

[2] R. Hirschfeld, P. Costanza, and O. Nierstrasz, "Context-oriented programming," *Journal of Object Technology*, vol. 7, no. 3, pp. 125–151, March 2008.

[3] B. Magableh and S. Barrett, "Pcoms: A component model for building context-dependent applications," in *Proceedings of the 2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, ser. COMPUTATIONWORLD '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 44–48.

[4] http://www.cisco.com, 2011, [Online; accessed 1-April-2011].

[5] Y. Chang, S. Peng, T. Wang, S. Chen, Y. Chen, and H. Chen, "Autonomous indoor wayfinding for individuals with cognitive impairments," *Journal of NeuroEngineering and Rehabilitation*, vol. 7, no. 1, p. 45, 2010.

[6] M. Wagner, R. Reichle, M. U. Khan, and K. Geihs, "Software development method for adaptive applications in ubiquitous computing environments," http://www.ist-music.eu/MUSIC/results/music-deliverables/, IST-MUSIC, Tech. Rep., Mar 2011, [Online; accessed 1-March-2011].

[7] M. U. Khan, "Unanticipated dynamic adaptation of mobile applications," Ph.D. dissertation, University of Kassel, Distributed Systems Group, Kassel, Germany, may 2010.

[8] N. Paspallis, "Middleware-based development of context-aware applications with reusable components," Ph.D. dissertation, University of Cyprus, Department of Computer Science, Nov 2009.

[9] A. Sindico and V. Grassi, "Model driven development of context aware software systems," in *International Workshop on Context-Oriented Programming*, ser. COP '09. New York, NY, USA: ACM, 2009, pp. 7:1–7:5.

[10] L. M. Daniele, L. Ferreira Pires, and M. Sinderen, "An mda-based approach for behaviour modelling of context-aware mobile applications," in *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications*, ser. ECMDA-FA '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 206–220.

[11] T. S. Parikh, "Using mobile phones for secure, distributed document processing in the developing world," *IEEE Pervasive Computing*, vol. 4, pp. 74–81, April 2005.

[12] "Object management group model driven architecture," October 2010, [Online; accessed 1-October-2010].

[13] "Enterprise collaboration architecture (eca) specification," pp. 1–202, Feb 2004.

[14] H. Hochmuller, "Towards the proper integration of extra-functional requirements," *Australasian Journal of Information Systems*, vol. 6, no. 2, pp. 98–117, 1999.

[15] E. Dashofy, H. Asuncion, S. Hendrickson, G. Suryanarayana, J. Georgas, and R. Taylor, "Archstudio 4: An architecture-based meta-modeling environment," in *Companion to the proceedings of the 29th International Conference on Software Engineering*, ser. ICSE COMPANION '07, 2007, pp. 67–68.