PCOMs: A Component Model for Building Context-Dependent Applications

Basel Magableh  
Department of Computer Science and Statistics  
Trinity College of Dublin  
Email:magableb@cs.tcd.ie

Stephen Barrett  
Department of Computer Science and Statistics  
Trinity College of Dublin  
Email:stephen.barrett@tcd.ie

Abstract—In this paper, we present a model to dynamically compose adaptable context-dependent applications using context conditions. The contributions of this work are: designing components as compositions of behaviour; embedding decision points in the component at development time to determine component behaviour; supporting reconfiguration of decision policies at runtime to adapt behaviours. The approach is a component-based one that modifies the application architecture by subdividing components into subsystems of static and dynamic elements. We map each context condition to a compositional template architectural configuration. Each context condition acts to select behavioural patterns, which combine to form application architectures. Patterns describe system adaptation in terms of adjustments of sub-elements of component behaviour. Our approach is dependent on the identification of sub-elements on the component suitable for localising context dependent changes such that alternatives versions tailored to specific context to be woven in the application at runtime as context varies. We illustrate this approach with a simple case study.

Index Terms—components model, Context-awareness middleware, context-dependent adaptation.

I. INTRODUCTION

It would be advantageous if mobile context-aware applications were able to adapt their application behaviour dynamically to varying contextual situations. Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves [1].

In order to gain such benefits, we argue that context-dependent applications should be built from composable architectural units. Our method is based on the assembly of applications from components that are context-unaware, but from a set that includes various alternative component implementations targeted for different contextual situations. Thus, the readying of an application in its totality for specific context is achieved through the assembly of the appropriate component implementations. This differs from the majority of contemporary work which seeks to embed awareness of context in the functional implementation of applications. This of course is dependent on an appropriate theoretical and practical basis for system decomposition along contextual lines, and a suitable mechanism for context aware software assembly and dynamic system reconstruction as context varies. In this paper, we outline such a scheme.

We base our method on the concept of a primitive component [2]. A model that deploys components as compositions of very fine-grained sub components. In this paper we illustrate this approach by way of an example video conferencing application, demonstrating the functional varying of features such as support for full-colour video, as context changes.

To make this concept feasible in practice, we introduce the PCOM model that supports the development of context-aware applications via three contributions. First, a development methodology to design and implement context-dependent applications. Second, a component framework design, describing the component structure and the composition mechanism. Third, a middleware design which is deployed between the (OS) layer and the context-aware applications that manipulate the adaptation process by changing the application architecture via the replacement of components associated with the varying context.

The rest of this paper is organised as follows. Section II, discusses Context Programming Techniques, middleware supports context-awareness systems and component-based systems that support adaptation. Section III, proposes the PCOM component framework, the component structure and the composition mechanism. Section IV describes the main components in the PCOM middleware supported by a runtime model. Section V illustrates an application architecture supported with a concrete example. Section VI provides the conclusions from this research. Future works are defined by Section VII.

II. RELATED WORK

A Context-oriented Programming (COP), proposed in [3], treats context explicitly and provides mechanisms to dynamically adapt behaviour in reaction to changes in context. This is possible even after system deployment at runtime. Motivation for using Context-Oriented Programming is that it directly supports context-dependent behavioural variations. Other motivation also includes the support of dynamic representation of layers and their scoped activation and deactivation in arbitrary places of the code, which is essential in COP. The middleware could activate or deactivate specific features dynamically by manipulating predefined variation points (Decision points) without modifying the original code. Those decision points predict the expected application behaviour in the component’s layer level.
The contribution of Meuter et al. [4] consists of a modelling approach, called Context-Oriented Domain Analysis (CODA), which is a systematic approach for gathering requirements of context-aware systems. The outcome from CODA model is to enforce software engineers to think of context-aware systems as pieces of basic context-unaware behaviour and Behavioural variations.

Many related middleware-based solutions that focus on context-aware system and component-based frameworks are proposed in the literature [5], [6], [7]. MADAM (Mobility and Adaptation-Enabling Middleware) aim to build adaptive applications for mobile devices using architecture models. The MADAM architecture provides a framework for developers to design component-based applications and support its execution in a distributed environment. MADAM requires an application to be divided into a hierarchical set of components.

Aspect-Oriented programming language extensions provide constructs for modularising crosscutting concerns. A crosscutting concern is some program behaviour, functional or non-functional, that does not dominate modularisation of a program. However the focus of feature-oriented programming is on compile-time selection and combination of feature variants. Considering aspects as reusable parts that can be woven and attached in the component is presented by [8], they provide a new component and aspect model, which defines components and aspects as first-class entity.

Several component-based deployment solutions have been specified in the literature such as CCM, EJB, .Net and J2EE. The process of deployment is not completely context-aware [9].

Pulnah et al. [10] Introduce a component model based on the difference between static and dynamic parts of the system's component. The possibility of embedding dynamic decision points (DP) at development time in the component enables any degree of dynamicity at runtime by supporting decision polices reconfiguration.

The PCOM component designed as compositions of behaviour; embedding decision points in the component at development time to determine component behaviour; supporting reconfiguration of decision policies at runtime to adapt behaviours. The PCOM model modifies the application architecture by subdividing components into subsystems of static and dynamic elements. We map each context condition to a composable template architectural configuration.

III. PCOM SOFTWARE COMPONENT MODEL
A. Problem Space

Component-based development aims to compose systems from prebuilt software components. Current software component models are not ideal to fulfil the requirements of context-dependent applications, they use components that are not easy to reuse and the composition mechanisms are not well defined and hard to apply systematically [11].

Composition can take place during different stages of a components' life cycles. 1) At the design phase, during which components are designed, defined and constructed in the source code, and possibly when they are compiled into binary code. 2) The deployment phase during which binaries of components are deployed into the target execution environment for the system under construction 3) At the runtime phase during which binaries of components are instantiated with data and these instances are executed in the running system [9].

The taxonomy of software component models reveals that no existing model has composition at runtime deals with context information, that suit context-dependent application adaptation. Current models also do not have well defined composition theory that supports systematic composition. In software component where components are objects. The composition of two objects by message passing is not supported by a composition theory and the result is not a single object but just two objects calling each other's methods. Architectural units compose via their ports but they do not have systematic composition even they have a simple composition theory.

The PCOM component model modifies the application architecture by subdividing components into subsystems of static and dynamic elements. We map each context condition to a composable template architectural configuration. Each context condition acts to select behavioural patterns, which combine to form application architectures. Patterns describe system adaptation in terms of adjustments of sub-elements of component behaviour. Our approach is dependent on the identification of sub-elements on the component suitable for localising context dependent changes such that alternatives versions tailored to specific context to be woven in the application at runtime as context varies.

B. PCOM Primitive Component Model

We call the PCOM component as any subpart of the software system that is associated with specific context condition. A PCOM component is a unit of behaviour contractually specified by interfaces and explicit dependencies (i.e. standard component model). A PCOM component structure is shown in Figure 1. The PCOM component has multiple ports and decision points.

Each port identifies a point of interaction between the component and its environment. A component may provide multiple interfaces by using different types of ports. A port can represent an interface as simple as a single procedure signature, or more complex interfaces, such as a collection of procedure calls that must be invoked in certain specified orders, or an event multi-cast interface point. The Decision Point identifies a point of interaction inside the component by activating or deactivating the component's layers.

The PCOM component consist of two parts: static behaviour (SB) and dynamic behaviour (DB) as in Figure 1. The component's static behaviour part represents the application behaviour which is not context-dependent. The component dynamic behaviour part is context-dependent functionality directly affected by the execution context, such as location or bandwidth level. A component’s dynamic parts have many layers. Each layers controls the activation or deactivation of more specific features associated with specific behaviour. The architecture description language (ADL) in Figure 1 shown the
Developers design one or more PCOM components to map specific context information. For example, the bandwidth status has three possible conditions (normal, low, or high). For each condition a PCOM component is designed. For the low bandwidth PCOM component zero or more Decision points are embedded. Decision points are manipulated by decision policies at runtime. Decision policy is a predefined description used to control the PCOM component's layers to manage the components' behaviours [10].

C. Context-aware systems assembly

Composition is a major issue in components frameworks, as components are used as building blocks or assembled together into large systems. A composition language is needed to define the composition. ADL and ADL-like are ideal composition languages [9]. Components are connected using three methods: direct connection, connectors, or patterns.

In the PCOM framework, we use a composition plan and design Patterns model. The composition plan recursively describes the composite components, and the connection between them by describing several design Patterns. A single composition plan could represent one possible realisation of the associated PCOM component. A design Pattern typically shows relationships and interactions between components’ dynamic behaviour parts. Also it describes the dependancy between components. The point from using composition plan and design patterns is the need to support two type of composition. External composition and Internal composition.

1) Internal composition: Internal composition describes the composition inside the component by replacing its implementation with another to satisfy specific dynamic behaviour. The dependancies among component can be described using design patterns. Components are bind together to satisfy their dependancies. The binding is performed at runtime and provides the necessary adjustments to introduce runtime behavioural variations. The specification of PCOM component is driven by a context oriented selection process.

2) External composition: External composition illustrates the mechanism to connect components via ports and connectors. A Component is modelled as requiring and/or providing services from other components to achieve specific behaviour. A connector uses the description illustrated by the composition plan to connect components A and B. As in ACME [12]. The Connector consists of two parts: interfaces and layout. The interface is used as input/output terminal between components. The layout describes the internal configuration of the connector. It shows the mechanism of connecting two interfaces internally. In the PCOM framework we use two type of configuration as in ACME [12], unidirectional connection (to) and bi-directional connection (and).

IV. PCOM MIDDLEWARE ARCHITECTURE

The structure of the Primitive middleware platform is shown in Figure 2. The context information is retrieved using physical context sensors, other context information such as blood pressure or body temperature are retrieved from hardware-based sensors (e.g. BIO-Sensors). The resources sensor retrieves resources information about bandwidth and battery. The OS sensor retrieves information by means of OS calls about CPU, memory, and disk space.

The context manager gathers context information from sensors. The context manager pushes its sensed conditions into the middleware to trigger the adaptation process. The decomposition component decompose application’s instance into PCOMs component, and Decision policies. The Decision Evaluation Module (DEM) retrieves the decision policies described by the developers at design time from the policy storage.
The PCOM framework manipulates components after the application recomposition has been designed. It uses the design pattern mentioned in the decision policy to configure the composition plan. The PCOM framework produces a new application’s architecture described by a composition plan. The adaptation manager uses that architecture with the composition plan to add new PCOM components or replaces old PCOM component to/from the application.

The configuration manager evaluates the modelled application’s instance. It verifies the application compatibility with the underlying resources and the expected quality of services. It checks if the application instance is providing the same services with the same quality as the older version did. If the verification process fails then it notifies the adaptation manager to retry the adaptation process.

A. Runtime Model

The runtime model outlined in Figure 3. Describes the adaptation process when context change is detected. The context model notifies the context manager about the context changes. The component manager identifies and selects the associated components which are involved in the current application from the PCOM component framework.

Using the decision points which are populated at runtime and the decision policies. The PCOM framework manipulates the selected PCOM components to produce an application’s architecture and a composition plan. The adaptation manager uses the composition plan to start constructing a new version of the application. Finally, the application is configured by the configuration manager.

V. CONTEXT-DEPENDENT APPLICATION EXAMPLE

A. PCOM Application Architecture Design

PCOM applications are composed of PCOM components that map abstract context conditions to. An application is modelled as a tree of context conditions, with PCOM components are the leaves of the tree. The application tree represent the dependencies between components. The components’ dependencies are described using design patterns.

Consider a simple video player application designed using the PCOM model from a set of PCOM components. In this example we are going to describe how the video player could changes its application behaviour depend on variant context conditions. The possible adaptations are: data richness, user interface presentation, application redeployment.

Data richness: Application react to the dynamically changing network connection level by adjusting the resolution and the colour-full depth of the video.

User interface presentation: Adjusting the screen brightness based on the battery level, quit unused applications, push unsaved data to a remote server.

Application redeployment: Redevelopment of application’s component to different device to convert audio interaction to text interaction by deploying speech-to-text component hosted in a remote server when the memory resource is low.

To manipulate the application possible behaviours. Developers design PCOM components that mapped to these context conditions (bandwidth, battery and memory). Layers and decision points are embedded in the PCOM components to support the adaptation. Developers must describe Decision policies at the design time to describe the components’ possible behaviour. The following decision policies are predefined and attached to the PCOM components.

Decision policy 1: if bandwidth drops to low, change the video resolution and switch from colourful to black and white video.

Decision policy 2: if low battery is detected, Adjust the screen brightness, switch off video streaming or audio streaming, quit unused running applications.

Decision policy 3: if the available memory drops to low, deploy speech-to-text PCOM component in remote server, push text messages as pop up window in mobile device.

Decision policy 4: if network connection is GPRS, battery is normal and memory is normal. redeploy the video streaming to a remote server.

B. Video Player Context-dependent Example

Scenario 1: The user is doing Skype call with audio and video streaming. Bandwidth drops dramatically to low. The Audio streaming is more important than video for the user to finish the call. In this case the middleware evaluates the predefined decision policies. Decision policy 1 fitness the current bandwidth condition (i.e. low bandwidth). The middleware uses the PCOM component’s layer to change the video resolution and the colourful depth to black and white.

Scenario 2: The user setting in his home streaming a video from a security camera installed in his shop. The mobile battery level drops to low. This will trigger the adaptation process. Decision policy 2 is selected by the middleware after evaluating the context. The middleware applies the policy by reducing the screen brightness, forcing unused applications to quit and Pushing unsaved data to a remote server.

Scenario 3: The user is doing Skype call with a friend discussing an important PDF file. Bandwidth is sufficiently fast and cheap. The available memory drops to low. The middleware trigger adaptation process, that gives the user the facility to continue reading the file and converting the audio interaction to text messages. The middleware deploys
a PCOM component in remote server, after selecting decision policy 3. The remote PCOM component convert the incoming audio streaming to a text messages. Then its pushes the text messages to the user mobile device. The user replies by text messages. The middleware pushes the outgoing text message to the remote server PCOM component, that converts the messages to an audio streaming. The user’s friend receives the text messages in form of audio interaction.

Scenario 4: The user switch from WLAN connection to GPRS, while streaming the video from the security camera. This trigger the middleware to select decision policy 4. The middleware redeploy the video streaming to the remote server as the network connection expansive and not efficient. The user could view the video after a sufficient and cheap network connection is available.

The level of application reconfiguration varies between the middleware mentioned in the state of art [5], [6], [7]. In the PCOM model reconfiguration accomplished by changing sub-elements inside the components related to the dynamic behaviours. In other middleware the adaptation accomplished by reconfiguring the application architecture entirely by adding or replacing components by others to satisfy the current condition.

VI. CONCLUSION

PCOM Model proposes new development methodology, a component framework, and a middleware. Building context-dependent system from PCOMs gives the application a high level of dynamicity to change their behaviour to several context conditions at runtime. The PCOM component model support dynamic mapping between components’ dynamic behaviour and context conditions.

Decision policies are provided at design time on the components level to simplify the adaptation process. The use of a composition plans as a planning-based adaptation strategy gives the middleware the ability to predict the expected behaviour of the application.

The PCOM middleware could be used in several fields of pervasive computing such embedded system, and context-dependent systems. The contributions of this work are: designing components as compositions of behaviour; embedding decision policies in the component at development time to determine component behaviour; supporting reconfiguration of policies at runtime to adapt behaviours. The approach is a component-based one that modifies the application architecture by subdividing components into subsystems of static and dynamic elements.

VII. FUTURE WORK

In this paper, we showed the general idea of a context model, component, and a middleware. A generic context requirements model is needed to analyse varying context conditions. Anticipating context conditions at an abstract level to map the PCOM is a challenge.

Policy configuration mechanisms are needed, the mechanisms explain how those policies are designed, processed, and configured to allow a new behaviour that was not initiated at design time. Finally to implement the PCOM component model an implementation language is required to specify the implementation process.

REFERENCES


