國立交通大學

資訊工程系

博 士 論 文

延伸性與可適性在行動執行環境的研究

A study of extensibility and adaptability for mobile environment

研 究 生：高子漢

指導教授：袁賢銘　教授

中 華 民 國 九 十 五 年 六 月

延伸性與可適性在行動執行環境的研究

A study of extensibility and adaptability for mobile environment

研 究 生：高子漢　　　Student : Tzu-Han Kao

指導教授：袁賢銘　　　Advisor : Shyan-Ming Yuan

國 立 交 通 大 學

資 訊 工 程 系

博 士 論 文

A Dissertation

Submitted to Department of Computer Science

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

in

Computer Science

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年六月

延 伸 性 與 可 適 性 在 行 動 執 行 環 境 的 研 究

學生：高子漢　　　　　　　　　　　　　指導教授：袁賢銘

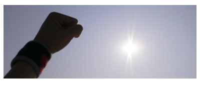
國立交通大學資訊工程學系（研究所）博士班

摘　　要

各種行動裝置日新月異。企業、娛樂、教育等應用都積極朝著隨時隨地透過任意的行動裝置提供服務。這一個論文主要目標就是『加速開發者在各種行動裝置上開發應用，讓使用者可以隨意更換不同設備，達到無所不在的使用情境。』

要達到此目的，我們針對幾種不同的行動執行環境。設計一 Framework: U-CAF (Ubiquitous Context Adaptation Framework)，並且設計三個系統實作。探討結合環境感知、元件調適以及 XML 轉化的技術達到此目標，

在三個系統實作 (CAAS, Ubi-Adapting, G^2)中。在環境感知部分，我們運用 W3C CC/PP 以及 WAP UAProf，以及 RFID、Bluetooth 等技術。藉此我們可知道使用者使用的設備以及環境資訊。呈現轉化模組：我們運用 XSLT/XPath 轉化技術，將介面同時轉化成多種不同行動裝置上的 Web 呈現語言…等等。在元件調適技術上，我們設計自動調適應用程式組成的演算法以及動態的重組應用程式結構的能力。結合這些技術，使得系統上的行動應用程式具自我順應能力。



以 UbiAdapting 為例，我們在系統中設計了 Adapting Service Container 以及 Personal Mobile Agents。在前者，除了感知環境的呈現轉化功能以及感知環境後的自我順應，系統開發人員或是服務的提拱者可以新增更多的感知環境服務。用以增加系統感知使用設備後的隨需應變功能。

此技術目標在提供程式開發以及自我順應，應用服務建構者、數位內容以及資訊提供者可以藉由不同的行動裝置，將資訊傳達給使用者。而使用者也可以就所在的環境使用合適的設備存取這些服務。我們實作了幾個應用，像是圖書館、電影查詢瀏覽，讓這些應用可以被自動地決定回傳給行動設備合適的內容。藉此，使用者可以更換各種行動設備享受無所不在的使用情境。

A study of extensibility and adaptability for mobile environment

Student: Tzu-Han Kao Advisor: Shyan-Ming Yuan

Department of Computer Science
National Chiao Tung University

ABSTRACT

Wearable, handheld, and embedded or standalone intelligent devices are becoming quite common and can support a diverse range of applications. In order to simplify development of applications which can adapt to a variety of mobile devices, we propose U-CAF (Ubiquitous Context Adaptation Framework). We want to provide Rapid-development, Adaptability, Flexibility for programmers, seamless-use for end-users, and Easy-upgrade for system developers in this framework. U-CAF contains API, PUML/PGML, SDK, context awareness, adaptation, and transformation mechanisms for application development on mobile execution environments. We implement three systems: CAAS, Ubi-Adapting, G^2 to realize our design. In the future, we expect U-CAF to be enhanced for further development of widespread applications for ubiquitous computing environments.

Acknowledgement

終於換我寫博士論文致謝了。這真的要感謝很多人，從家人到朋友，從學校的師長、學長、學弟妹到在這人生努力過程的伙伴們：民視、台灣 IBM、資策會、工研院…，以及我參加過每一個活動，每一場比賽一起努力的學弟妹們，以及各個參賽與活動的伙伴們。

在博士學位的修業過程中，父親 高進發先生、母親 劉幸珠女士的鼓勵與支持，是我源源不絕的修業動力。記得在 2005 年暑假，一次去病房探望父親，在病床上問我哪時候畢業，父親的眼裡泛著淚光。在當下的我也感到相當的不孝。而母親總是在疲憊的工作之餘還有力氣關心在學校的我。真的很感謝你們的諄諄教誨。我想可以拿到博士學位最開心的應該就是我的父母親。也期許自己可以有一番作為，法天地，作『正』事以報答父母之恩。

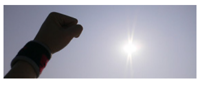
在修業的過程中，不可或缺的角色我的哥哥 高子龍、以及我的弟弟 高子璽。很感謝你們幫忙我修改論文投稿的英文文句。我還記得在 2003 年暑假時候，跟我哥哥約一天中午在園區的一家餐廳修改論文，那年暑假短短的三個月我上了一篇 Computer Science & Interface (CSI)的期刊論文。也奠定了論文累積點數的基石。弟弟個性也是相當堅強，一個人到嘉義求學唸書，心理所想是自己工作可以自給自足。在論文的寫作過程，弟弟每每不厭其煩的修改我的論文，也常糾正我的英文謬誤。一篇論文修改後又接著一篇論文。而老弟也會我在洩氣時給我莫大的肯定。很感謝我的兄弟在這研究所研讀其間的親情與支持。

交通大學資訊科學系分散式系統實驗室-- 我的研究所求學的實驗室。94 學年度起，交通大學資訊學院成立，資訊科學系與資訊工程系合併。現稱交通大學資訊工程系資訊科學與工程研究所分散式系統實驗室。我的指導教授 袁賢銘 教授，同時也是這一個實驗室的主持人。老師的作風開放，而在這實驗室我也有很大的發揮空間。在碩一、博一、博二、博三…一直到了博五。一路走來，跌跌撞

撞。感謝老師六年的指導以及在這實驗室也有機會指導學弟們。

在歷屆實驗室的學長、學弟妹。也是我攻讀博士論文不可或缺的角色。2000 年剛進實驗室，那時候的我的想法自卑而且也偏保守。很感謝當時鼓勵指導我的學長 焦信達 學長、許瑞愷 學長、劉旨峰 學長，可以說是在碩士班時期給我很多指導起及期許。如今 2006 年，學長們有已經結婚或有擔任國立大學的教授職位。這也是我感到開心的地方。而伴隨著我在研究所修業期間的實驗室學長，就是 葉秉哲 學長、鄭明俊 學長、林獻堂 學長、蕭存喻 學長、邱繼宏 學長、以及 吳瑞祥 學長。秉哲學長不但學問技術高而且有著一個熱心助人的心，這也是我學習效法的對象。學長也蠻關心我，除了在讀書會上的支持，在論文口試前後的關切以及鼓勵。明俊學長，記得在碩研究所修業早期，常常有一些煩惱，也都是找他聊天。繼宏學長，他也是一位很熱情的人，總是有很多點子在處理事情上。存喻學長、瑞祥學長，在他們身上可以學到把計畫做好的方法。很感謝在那時候大家在一起的聚餐去竹北吃東西的…等等歡樂時光、以及在攻讀期間的關心與鼓勵。在他們身上我學習到了熱情、熱誠、技術、以及待人處事。

早期在碩一到博三期間，跟我一起實驗室的同學 周宜興、洪傳寶、以及 林均翰。還有研究所 90 級學弟妹：林書慶 學弟、黃郁芳 學妹、姚立三 學弟、李宣鋒 學弟；研究所 91 級學弟妹：沈聖博 學弟、洪崇凱 學弟、蔡明耀 學弟、劉昀昇 學弟、陸振恩 學弟、蘇科旭 學弟；92 級學弟妹：顏志明 學弟、沈上謙 學弟、於之均 學弟、林建豪 學弟、葉倫武 學弟、林慧雯 學妹以及 朱文如。很懷念那時候大家在一起的時光。也謝謝你們在人生道路上的幫忙。研究所 93 級學弟們：陳勇宇 學弟、蔡紀暘 學弟、謝偉德 學弟、林家鋒 學弟、林良彥 學弟、陳俊元 學弟、范志歆 學弟、李杰叢 學弟。謝謝大家在實驗室的相互幫忙。紀暘 學弟與 勇宇 學弟跟我一起參加『教育部嵌入式軟體設計競賽』。也感謝學弟們的有心。下面這文句是我在參加完幾個比賽之後的感悟。



比賽是一起努力的過程

偉德 學弟有一台很不錯的車，有時候也都麻煩 偉德。家鋒 學弟很謝謝你介紹讓我認識交大領袖社以及交大禪學社。也謝謝在校內口試以及校外口試的幫忙。在我研究所研讀中期與我合作就是工研院電通所 W100 的鄭博文 課長、楊淑芬學姐、李卓俊 先生。

在攻讀博士的後期，那時候可以說是按耐不住自己想要想前衝的動力，同時也想鍛鍊自己。在 2004 年底有機會認識寰震科技 林克仁 董事長、董事長特助曾能壯先生、林耀珍 副總。並且與寰震伙伴以及友人 邱銘彰一起組織讀書會。也感謝 王子彥、孫文駿、叢培侃、Jesse 的加油以及讀書會伙伴在這期間的鼓勵。當然也感謝有良會到資策會作系統簡報，並認識資策會網多所 馮明惠 主任；以及畢業前夕的工研院電通所 P 組的伙伴們。謝謝大家的幫忙與加油。

此同時，我也參加了大大小小的比賽，由北到南，由台灣到兩岸三地 (教育部九十二學年度通訊專題製作競賽、2004 打造無限夢想行動家- 2004 通訊大賽、2005 Microsoft Image CUP、教育部嵌入式軟體設計競賽、第一屆東森提案競賽、以及 IBM 在兩岸三地所舉辦的『第二屆 IBM 杯高校校園創新設計大賽』。很感謝在競賽中以起努力的伙伴們。在 2005 Microsoft Image CUP 的原資科 95 蔡宗翰 學弟、林瑋璿 學弟、周鴻仁 學弟。感謝大家在競賽前一天夜晚的一起努力，要謝謝 瑋璿父親的開車幫忙。

在第二屆 IBM 杯高校校園創新設計大賽，是我畢業前夕所參加的最後一場比賽。要感謝的主要有一同參賽的學弟們、在這比賽中幫忙的伙伴、以及賽後相互鼓勵加油的伙伴。逢甲資工 楊晉安 學弟，是一位有責任心、開朗、有學習熱誠的學弟。我跟他是在 2005Microsoft Image CUP 認識，更一起參加 IBM 競賽的伙伴。一起挺進北京決賽的還有我很喜歡的學弟們：資工 97 李柏明 學弟、原資科 97 黃照展 學弟、資工 97 黃銘祥 學弟。也感謝幫忙的交大運管 97 李翼帆 學

弟。此外在此競賽過程，也感謝民視電視公司 侯銘罡 組長、楊國宏 先生以及奈訊科技的 傅振倫 經理、陳建隆 經理的設備借用。更有台灣IBM的 吳明峰 經理、李敏宏 經理的鼓勵。以及台灣IBM前總經理 許朱勝 先生的道賀。也感謝多位大中華地區的伙伴們，北京IBM的 湯利華 小姐的幫忙。賽後與各隊的伙伴認識，甘肅蘭州大學的 程廣輝 學弟，南京南開大學的 王鋒 學弟、韓坪良 學弟，北京大學 秦辰 學弟還有很多大中華的好手們。廣輝也跟我一起共事到現在，大家彼此相互加油鼓勵。在競賽的過程中，一步一步，從顛頗開始到學習如何團隊合作以及領導組織，也在每一個過程中與大家分享、關懷。下面這文句是在 2006 年 5 月 1 日寫下，也與大家共勉之

生命的精彩就是可以不斷的創造價值

而一段精彩就在於可以大家一起分享感動

在 2006 年初最後寫論文期間，也是畢業最為關鍵的時間。很感謝交大機械 97 曹育銘 學弟的開朗大小孩個性，彼此相互勉勵。他一步一步往上爬，年紀輕但是有著一顆熱心以及助人心。感謝很多這時候認識的友人 王常威 學弟、劉駿 先生。謝謝 常威介紹好吃的東西，也謝謝 劉駿先生介紹讀書會以及伙伴給我認識。還有很多我非常喜歡的學弟妹們。謝謝交大領袖社以及交大禪學社的伙伴立欣 學妹、陳立先 學弟、黃貴笠 學弟、林柏伽 學弟、以及李孟樵 學弟、陳憲磐 學弟、傅崇豪 學弟的加油。清大創研社的張仲瑋 學弟、施維濤 學弟、凱嶸 學弟、廖士霆 學弟、吳蘇容 學妹、李威寬 學弟、王孟倫 學弟、江承哲 學弟、傅彥鈞 學弟、林冠志 學弟、莊欣儒 學妹。還有很多清大的學弟們：盧鈞鈺 學弟、趙茂成 學弟、蕭百亨 學弟、陳松裕 學弟。團契的伙伴們：陳威霖 學弟、黃語慧 學妹、純如姐、丁瑩潔 學妹、焜哥、品祥、京荃、金亮、慧賢、韋恩、曉雯姐、游思遠的祝福與鼓勵。交大鍾鐸社的伙伴：陳彥堂 學弟、林聖瀚 學弟、

鍾欣儒 學妹、潘姒婷 學妹、吳祉嫻 學妹、李柏逸 學弟。還有系上的學弟妹們：黃宏文 學弟、朱峰儀 學弟、陳奕全 學弟、田燦榮 學弟、鄭偉升 學弟、林宣佑 學弟、蔡雅竹 學妹、林俊碩 學弟、林冠儒 學弟、辜博熙 學弟、陳健文 學弟、許文峰 學弟、許裕明 學弟、陳力行 學弟、以及 郭書庭 學弟，感謝這些學弟妹們的加油。沒有大家的加油，我想我的論文可能會遙遙無期。

也謝謝我永遠的好朋友們，國中同學 黃天保、陳建雄 的鼓勵。我永遠記得在台北捷運車站，兩位好友對於我畢業的期勉。以及高中一直到現在的朋友 陳勇全、李孟坤、蔡秉宏、以及 李建泓。也謝謝你們來訪參加我在交大博士畢業典禮。謝謝在研究所修業期間鼓舞的大學好朋友 張世昭、王景新、賴駿昇、以及 蔣佳宏。

謝謝在我最後論文口試，陪伴我的委員們：曾憲雄 老師、陳俊穎 老師、施仁忠 老師、郭譽申 老師、游寶達 老師、鄭憲忠 老師、張玉山 學長。感謝師長的寶貴的建議。也勤勉自己更加精進堅實。

我以 2006 4/2 寫下的這段文做為此篇致謝文的結尾。歷史的顏色我們一起走過，喜悅與分享，因為有大家。我很喜歡大家。也願與大家一起分享天籟、與自然齊鳴、而與天地共舞。

『輸』的時候要撐的住

施振榮

這有多少人可以作得到呢..？
超越 精進 反省 再出發 調整 掌握
在於開創、在於維持、在於可以『忍』
我的心，我不知道這是一個什麼情況，在幾年後回想起來。
但我知道這是一個『拓荒』過程

高 子 漢

交通大學 電資院 資訊工程系
資訊科學與工程研究所
2006 7/13

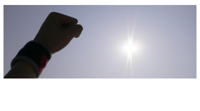


Table of Contents

Acknowledgement 5
Table of Contents 10
List of Figures 12
List of Tables 14
Terminology 15
Chapter 1 Introduction 16
    1.1 The environment 18
    1.2 The problems 19
    1.3   Objectives 20
    1.4 Application Model 21
        1.4.1 The programming model 21
        1.4.2 In an system implementation 22
        1.4.3 PUML/PGML and APIs 23
    1.5 Solutions 24
Chapter 2 Context adaptation 27
    2.1 Context-awareness 27
    2.2 Context adaptation process 31
        2.2.1 Application structure 31
        2.2.2 Attribute-based component decision algorithm 34
        2.2.3 Application Structure & Component Constraints 42
Chapter 3 PUML/PGML Transformation 45
    3.1 Pervasive User Interface Markup Language (PUML) 45
        3.1.1 Conceptual view 45
        3.1.2 The language description 47
        3.1.3 The elements of PUML 49
    3.2 Pervasive LoGic Markup Language (PGML) 53
        3.2.1 Design Principle 53
        3.2.2 The design of PGML 54
    3.3 Leveraging XSLT/XPath Transformation 60
        3.3.1 An overview of language transformation mechanism 60
        3.3.2 XSLT transformation procedure 61
    3.4 The use of combination of context adaptation and XSLT transformation 67
Chapter 4 System implementation 68
    4.1 Context-Aware Adaptation Service (CAAS) 68
        4.1.1 Example scenario 68

4.1.2 System overview 69
4.1.3 System architecture 70
4.1.4 Agent migration 77
4.2 Ubi-Adapting 93
4.3 Gateway of Gateway (G^2) 95
4.4 Applications 95
Chapter 5 Application development 95
5.1 PUML/PGML writing 95
5.1.1 Hand coding 95
5.1.2 WYSIWYG 95
5.1.3 The generated code 95
5.2 Application component and agent development 95
Chapter 6 Related work 95
Chapter 7 Conclusion and Future Work 95

List of Figures

Fig. 1 Form-based applications: the focused applications 17
Fig. 2 Device capability diversity 19
Fig. 3 The problems 20
Fig. 4 The application model 21
Fig. 5 A system implementation 22
Fig. 6 The languages and APIs 23
Fig. 7 The solution we propose 26
Fig. 8. A segmentation of Nokia 8310's WAP UAProf profile 28
Fig. 9 RFID application 30
Fig. 10. The back-end module of the application Image Gathering 32
Fig. 11. The structure of the application ImageGathering 33
Fig. 12 A component decision tree and its linked components 36
Fig. 13 An instance of a component decision tree on the right-hand side, and the associated components of the application ImageGathering on the left-hand side 38
Fig. 14 The implementation (linking list) of a decision tree 40
Fig. 15. The linking list of the decision tree illustrated in Figure 13 41
Fig. 16. The conceptual view of PUML 47
Fig. 17. A tree view of the XML Schema of PGML 55
Fig. 18 the XML Schema description of the <add> element 57
Fig. 19. The processing flow of PUML/PGML documents 60
Fig. 20 Integrating transformation and adaptation 67
Fig. 21. An overview of the system infrastructure 70
Fig. 22. The inner architecture of the context-aware adaptation framework 71
Fig. 23 Classes of component manager, component factory, and application context 75
Fig. 24. State transition diagram of the personal agent 77
Fig. 25. An internal view of a personal agent and an application structure 79
Fig. 26. Cases of users' movements 80
Fig. 27. The sequence diagram of ACPS 82
Fig. 28 The sequence diagram of ACPS with agent migration 83
Fig. 29 The sequence diagram of ACPS without agent migration 83
Fig. 30. The structure of the application ImageGathering by using three strategies. 85
Fig. 31. Agent migration using HAM 86
Fig. 32 Agent migration using FAM 88
Fig. 33 Agent migration using LAM 90
Fig. 34 The experimental setting 92

Fig. 35 The experimental results on HAM, FAM, LAM-B, and LAM-W 93
Fig. 36 The architecture of Ubi-Adapting 94
Fig. 37 the tested devices 94
Fig. 38 the overview of the G^2 95
Fig. 39 PUML/PGML & agent development 95
Fig. 40 the steps for development 95
Fig. 41 using the toolkit for development 95
Fig. 42 the toolkit we designed 95
Fig. 43 The top figures are the simulating results of the WML code, and the bottom is the result of J2ME code generated from the code of PUML/PGML 95
Fig. 44 The class diagram of programming agents and back-end modules 95
Fig. 45 a generic model 95
Fig. 46 an overview of G^{2}MR 95
Fig. 47 the components of G^{2}MR 95

List of Tables

TAB. 1 THE RELATIONSHIP OF SOLUTION AND OBJECTIVE 25
TAB. 2 THE TYPES OF CONTEXT INFORMATION 27
TAB. 3 THREE CATEGORIES OF COMPONENTS 33
TAB. 4 THE RESULT OF THE FIRST FIVE STEPS OF TRANSFORMING THE PGML DOCUMENT INTO THE J2ME MIDP CODE 64
TAB. 5. THE MAPPINGS FROM PUML TAGS INTO MIDP EXPRESSIONS IN THE PUML-TO-J2ME TRANSFORMATION STYLESHEET 66
TAB. 6. THE MAPPINGS FROM PUML TAGS INTO WML EXPRESSIONS IN THE PUML-TO-WML TRANSFORMATION STYLESHEET 66
TAB. 7. THE RESULTS OF THE FIRST MEASUREMENT 92
TAB. 8. THE RESULTS OF THE SECOND MEASUREMENT 92
TAB. 9 THE REVOLUTION 95
TAB. 10 THE APPLICATION DEMOS 95
TAB. 11 THE RELATED WORK 95
TAB. 12 THE SYSTEM WE IMPLEMENT AND THE FUTURE WORK 95

Terminology

Name	Description	Page
form-based application	Component an object, like java class in implementation	17
component	An object has some computational logic e.g. Java Object	33
function	An container can contain one or more components, and each component can implement the function	32
implement	A component of the component in a function provide computational logic for the function	32
application	An container contains at least one function	32
agent	An object with some state can carry at least one function	34
the front-end module	The front-end module is the part running on the mobile client side, such as WML, CHTML, etc.	31
the back-end module	The back-end module is applications carried by agents	31
ASCC	Application Structure and Component Constraints An application profile description.	35
context adaptation:	To adapt applications depending on users' context	31
mobile execution environment	WAP, J2ME, PersonalJava, and Microsoft CLI.	18
PUML	Pervasive User-interface Markup Language An XML-based UI description language	45
PGML	Pervasive logic Markup Language An XML-based logic description language	45
U-CAF	Ubiquitous Context Adaptation Framework The framework we design in this dissertation	25
CAAS	Context-Aware Adaptation Service A system we realize	68
Ubi-Adapting	A system we realize	93
G^2	Gateway of gateway A system we realize	95
G^2MR	Gateway of gateway with MHP and RFID A system we realize	118
WYSIWYG	What You See is What You Get A function of toolkit	103
follow-me	Application capable of following users	111

Chapter 1

Introduction

With the progress of mobile technology, embedded systems and information appliances have been developed; and various kinds of handsets, networked facilities, and personal mobile devices enrich our lives. These technologies have been applied to many fields. For example, there are networked TVs and home entertainment facilities in home appliances; internet-capable PDAs, mobile phones, wearable computers in personal mobile applications; and embedded servers in business applications. Accordingly, context-aware applications, which adapt their behaviors to a changing environment [1, 2] according to the context, such as indoor position, time of the day, nearby equipment, and user activities [3], can be developed. Context-aware mobile tourist guides [4] and location-aware shopping assistants [5] are two examples.

We can now foresee a ubiquitous computing environment [6] where a user can retrieve his personal information through any nearby computing facility, such as mobile and embedded computing devices, desktop computers, etc. In such an environment, information presented on the devices can be adjusted according to the context of these devices. One of the applications, called ImageGathering, where a multimedia campus guidance system is built on a campus, can be taken as an example. Wherever they are on campus, students can always inquire this system for the location of a building by using a Java phone, PDA, or a laptop. Depending on the context of the student's device, a formatted image suitable for the student's device can be delivered to the student. When a visitor would like to enter some building on the campus, he can use his Java phone for more information on that building, and then a

PNG image of 64x54 pixels will be sent to him. A notebook user can get a JEPG image of 340x256 pixels.

The delivery of the required image, depending on the context of device capabilities and user preferences, dominates the functions of this ImageGathering system. In addition, whatever device is used, users' applications will still continue. A user, for example, can use a desktop computer to check his daily report. When he moves from room to room, information about his report can still be acquired by a handheld PDA. In brief, we aim at providing a context-aware adaptive framework that can not only adapt functions of applications which personally rely on the context of the devices used, but also keep the executing states of applications even by using different devices. In this research, we focus on form-based applications, shown as Figure 1. the reasons are the screen of the client devices are small and the form capable of interaction with users.

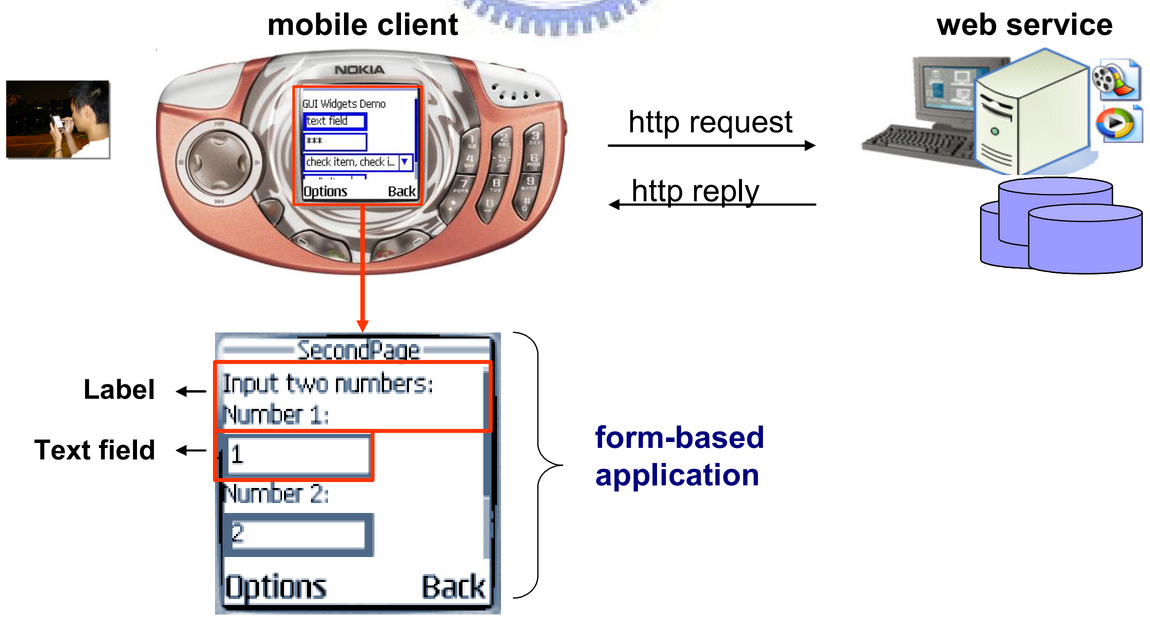


Fig. 1 Form-based applications: the focused applications

1.1 The environment

Mobile and wireless technologies have been changing over the past few years. Through mobile and embedded devices, such as PDAs, palms, smart phones, and Java phones, people can surf the content on the Internet. Besides, they can download and install applications from a content provider's server over the Internet, like Java game download. Currently there have been four kinds of the mobile execution environments on plentiful mobile appliances. These environments include WAP [7], J2ME [8][9], PersonalJava [10], and Microsoft CLI. In the Microsoft .NET platform, the mobile runtime environment supported can be classified into: (1) ASP .NET Mobile Pages [11] and (2) .NET Compact Framework [12]. The former attempts to support major PDAs, cell phones, pagers, and other devices, while the latter surpports all equipments running Pocket PC 2000, Pocket PC 2002, Pocket PC Phone Edition, etc. Called Mobile Execution Environments (MExE) in the standard [13], these environments—classmark 1 to 4—are defined by the 3GPP working groups. They stand for WAP, J2ME, PJava, and Microsoft CLI, respectively (see Figure 1). In short, device capabilities are diversity.

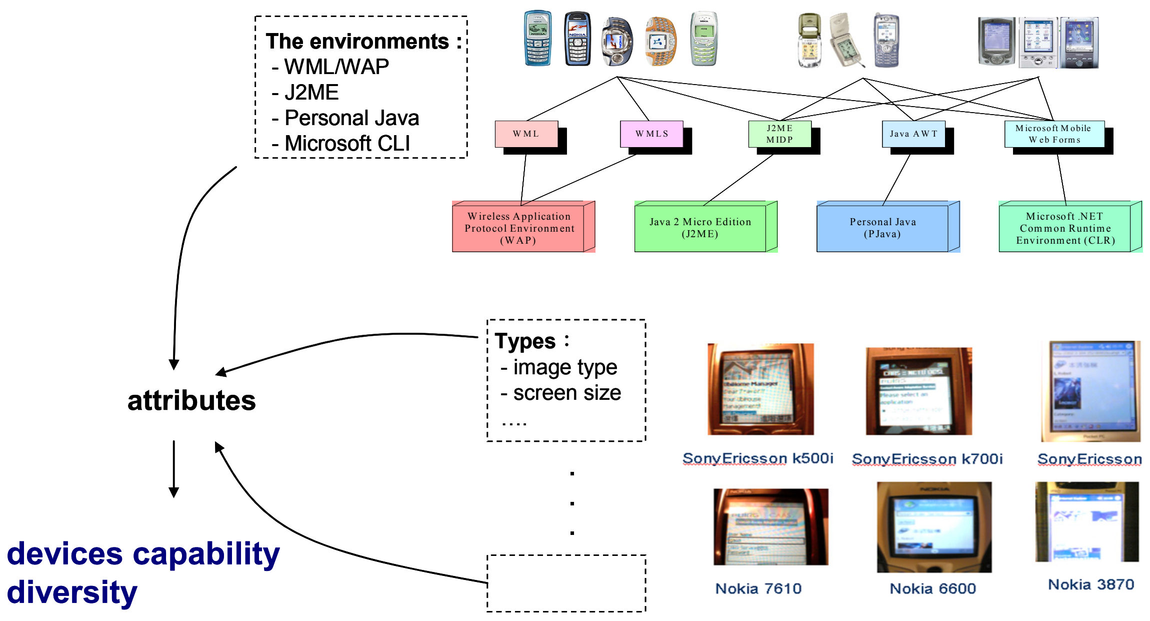


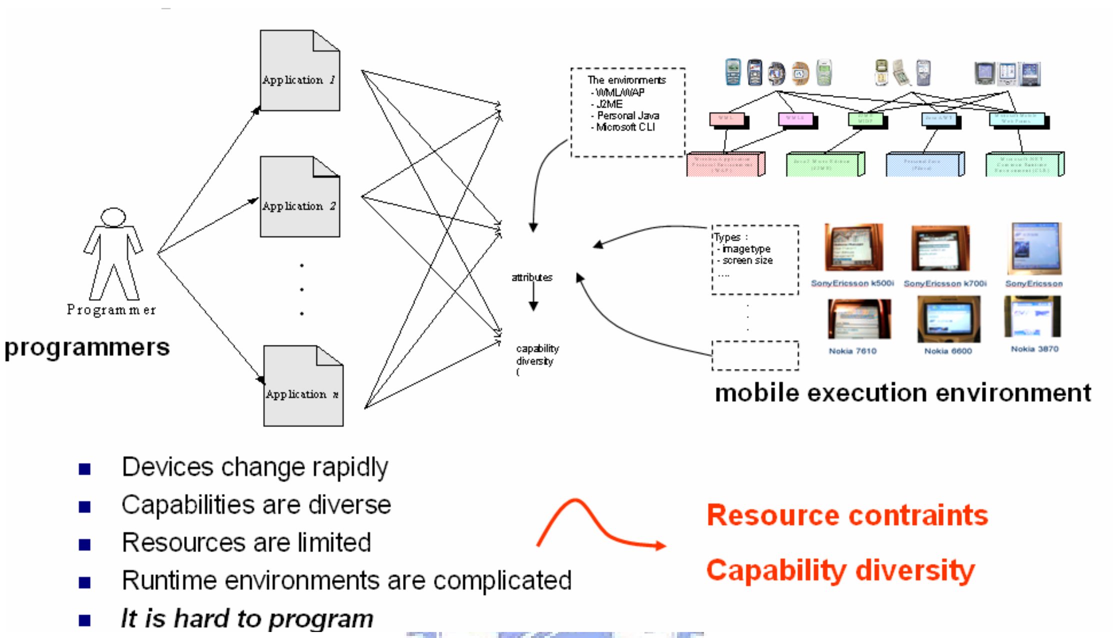
Fig. 2 Device capability diversity

1.2 The problems

Each of these mobile runtime environments has abundant resources and application interfaces (APIs) for application developing. Nevertheless, in application developers' points of view, developing applications for certain platform of the execution environments, they difficultly execute these applications on any other of the mobile execution environments. For example, a J2ME application cannot be executed on a cellular phone merely with a WAE platform (the runtime environment of WAP). Writing code of an application for each different platform is not economy.

Our research aims to achieve the objective: applications can be designed without concerning about what kind of the target mobile devices belongs to. In order to achieve goal, we attempt to exploit transformation mechanism to convert a program in some programming language into a program in another language. However, the programs, written in a language, hold the characteristics of that language. To transform the code into another language is

complicated. For example, C++ has the property of multiple inheritance, but Java merely has the property of single inherence. However, there two main problems: Resource Constraints and Capability diversity are illustrated in Figure 3.



- Devices change rapidly
- Capabilities are diverse
- Resources are limited
- Runtime environments are complicated
- *It is hard to program*

Fig. 3 The problems

1.3 Objectives

For programmers, we want provide Rapid-development, Adaptability, Flexibility. We accelerate application development on several mobile execution environments. Rapid-development means programmers can develop applications soon. Adaptability means functions of applications are adapted rely on the context of the devices used by the context-aware adaptive framework. Flexibility means we want to provide the framework of which the migration behavior of mobile agents can be chosen by programmers. For end-users, seamless-use means a user can change another device to use and the executing status for this user still continue, even the capabilities of the

used device differ from the original one. For system developers, we want to provide Easy-upgrade. The system can be upgraded soon, while new devices come.

1.4 Application Model

1.4.1 The programming model

We divide the application into two parts: the front-end module and back-end module. The front-end module contains two main constituents: an image display that can show images, and a requester that can send requests and receive the replied images. This module executes on the client device. The back-end module consists several constituents.

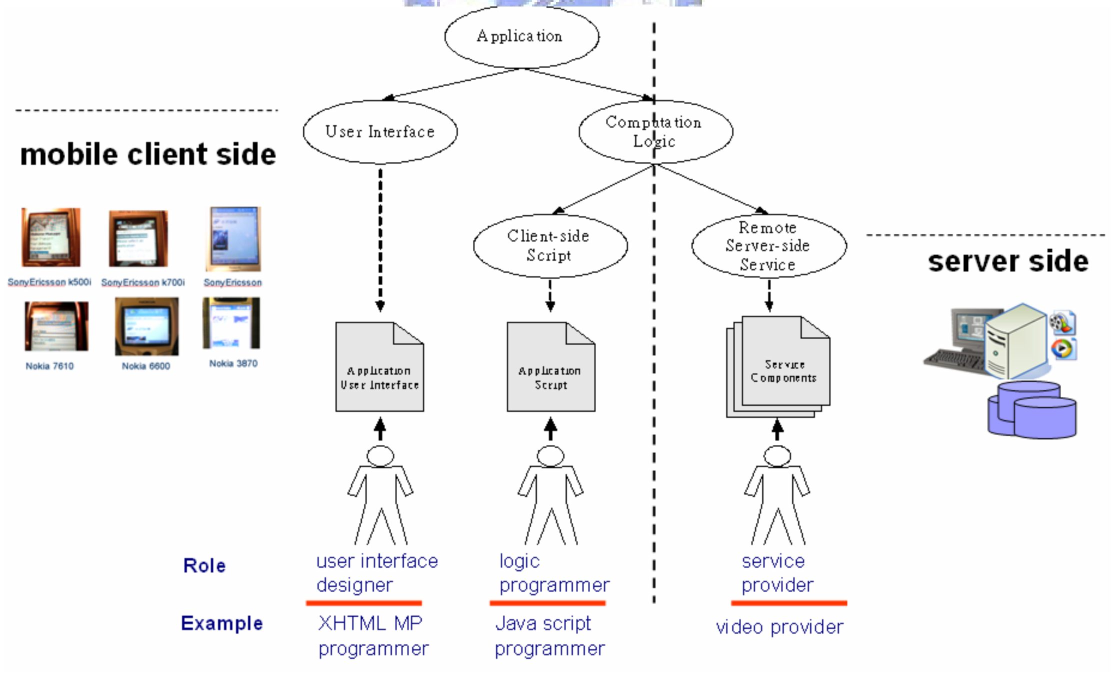


Fig. 4 The application model

1.4.2 In an system implementation

As shown in Figure 5, it illustrates Ubi-Adapting, a system implementation. From the bottom to top, we can there are five modules implemented. They are Application Profile, Context Profile, Context Awareness Module, Personal Agents, and Adapting Service Module. Application Profile, Context Profile, and Context Awareness Module are used in the context-awareness process of context adaptation. Adapting Service Module can contain one of performers to perform some action, such as application adaptation service and representation transformation service. On the top of the five components, there are server side components of applications. In the mobile client side, there are client modules in these environments: CHTML, personal Java, JavaME, etc.

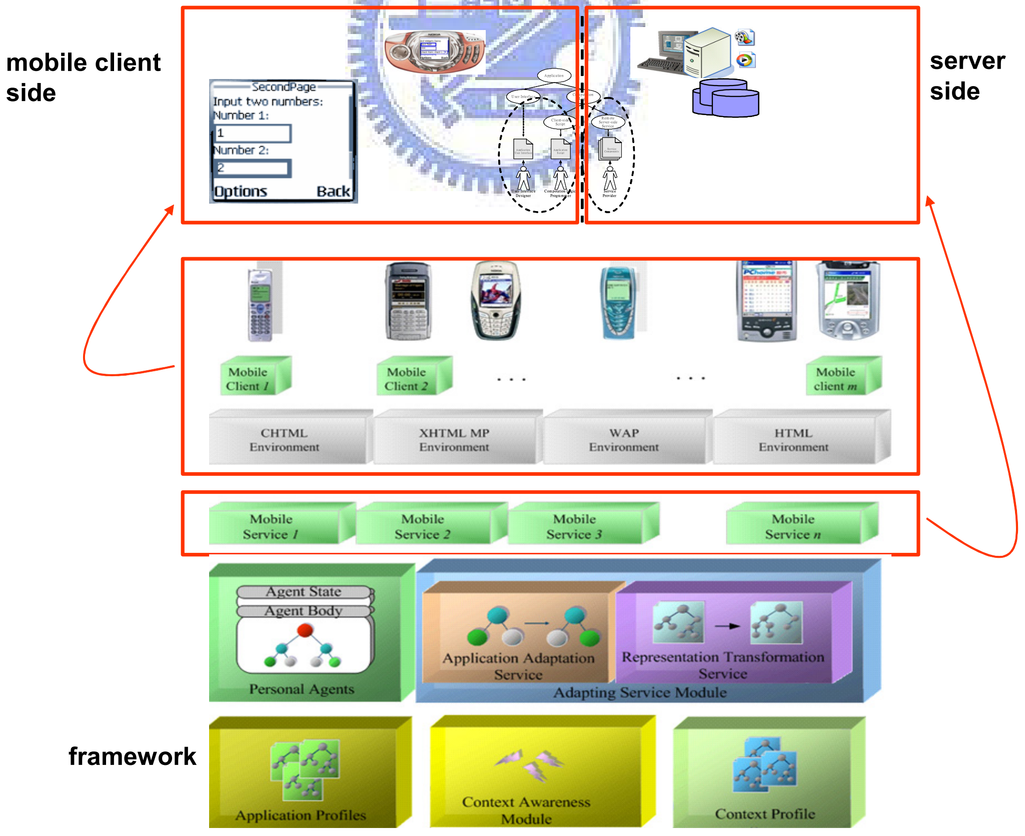
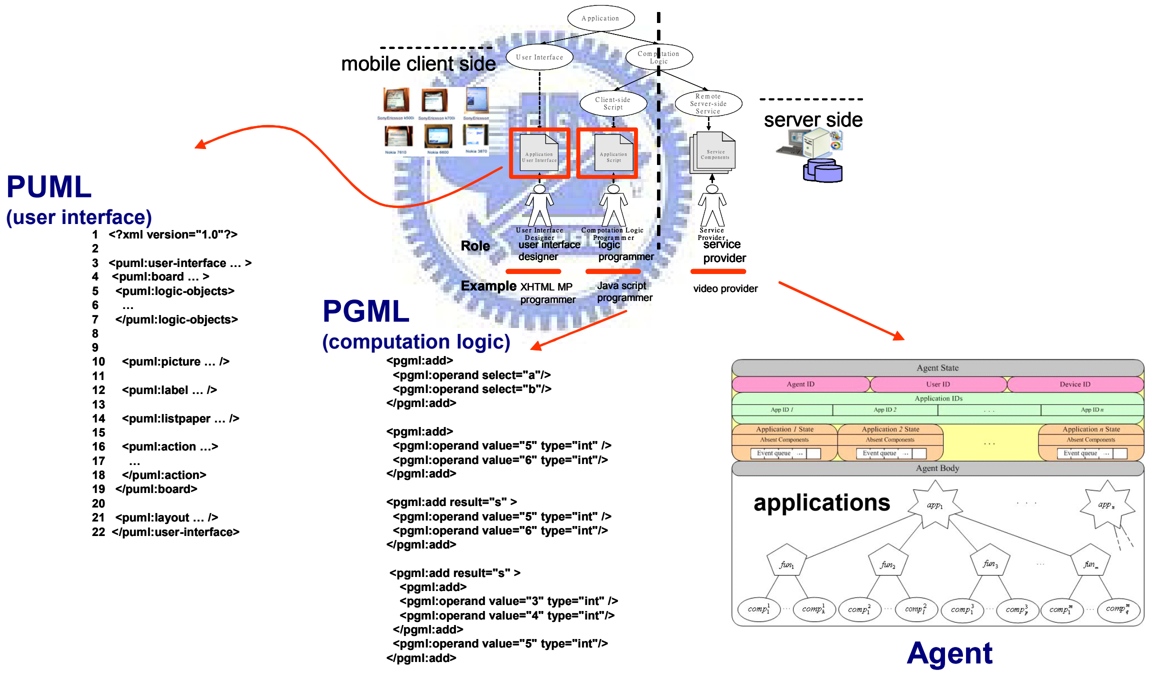


Fig. 5 A system implementation

1.4.3 PUML/PGML and APIs

In our programming model (Figure 6), we can use Pervasive User-interface Markup Language (PUML) [14] and Pervasive loGic Markup Language (PGML) [14] to write the code for the front-end module in the mobile client side. Several kinds of code which can be run on the client environments will be generated. In the server side, programmers can use API to construct agent and then to carry the server sides components. In this way, agents can carrying the server side components and migrate between different computers.



```
<?xml version="1.0"?>

<puml:user-interface … >
 <puml:board … >
  <puml:logic-objects>
    …
  </puml:logic-objects>


    <puml:picture … />

    <puml:label … />

    <puml:listpaper … />

    <puml:action …>
      …
    </puml:action>
  </puml:board>

  <puml:layout … />
 </puml:user-interface>
```

```
<pgml:add>
  <pgml:operand select="a"/>
  <pgml:operand select="b"/>
</pgml:add>

<pgml:add>
  <pgml:operand value="5" type="int" />
  <pgml:operand value="6" type="int"/>
</pgml:add>

<pgml:add result="s" >
  <pgml:operand value="5" type="int" />
  <pgml:operand value="6" type="int"/>
</pgml:add>

 <pgml:add result="s" >
   <pgml:add>
   <pgml:operand value="3" type="int" />
   <pgml:operand value="4" type="int"/>
  </pgml:add>
  <pgml:operand value="5" type="int"/>
</pgml:add>
```

Fig. 6 The languages and APIs

1.5 Solutions

We design U-CAF, Ubiquitous Context Adaptation Framework. Figure 7 illustrate this framework. It is revised from X-CAF [15]. Ubiquitous Context Adaptation Framework (U-CAF) contains software development kit and application programming interface, and service components. In this framework, we design context adaptation PUML/PGML translation, agent migration to approach our objectives.

***SDKs and APIs**:*

We design mobile designer, which is a GUI-based UI authoring toolkit with the visualized fashioning functions using drag-and-drop and WYSIWYG (What You See Is What You Get) measures to accelerate the development of user interfaces of applications on mobile execution environments. Currently, we have designed APIs for the development of the front-end module (Figure 7) and the back-end module (Figure 7).

Context adaptation:

The use of the context profile to decide proper components [16][17] in application adaptation is a critical issue. In this paper, we aim to develop an attribute-based algorithm that decides components appropriately by using dynamic and static context information (CC/PP and WAP UAProf profiles). These applied information will be explained in Chapter 2.

PUML/PGML transformation:

One is named Pervasive User interface Markup Language (PUML), which can describe user interfaces for applications on the small devices; the other is named Pervasive loGic Markup Language (PGML) to represent the computational logic of

applications.

Agent migration:

As a result, certain components of an application with heavy computing load can be enveloped into the back-end module and run at the server side. For follow-me, a personal agent [16][17], which can not only be anchored at a certain server to serve its owner, but also can carry back-end modules of applications as migrating among servers with its owner. This provides the flexibility for application developments.

Section 1.3 introduces the means of the objectives. The following table shows the relationship of objectives and solutions. To approach our objectives, we attempt the technologies introduced above. The following table shows the methods we design. These techniques will be explained in the later sections.

Tab. 1 the relationship of solution and objective

objective **Solution**	**Rapid Development**	**Adaptability**	**Flexibility**	**Seamless-use**	**System Easy-upgrade**
SDK and API	GUI toolkit & API				
Context awareness	W3C CC/PP WAP UAProf				W3C CC/PP WAP UAProf
Adaptation		Context adaptation		Context adaptation	
Transformation		PUML/PGML		PUML/PGML	PUML/PGML Transformation Stylesheet
Agent Migration			FAM, HAM, LAM		

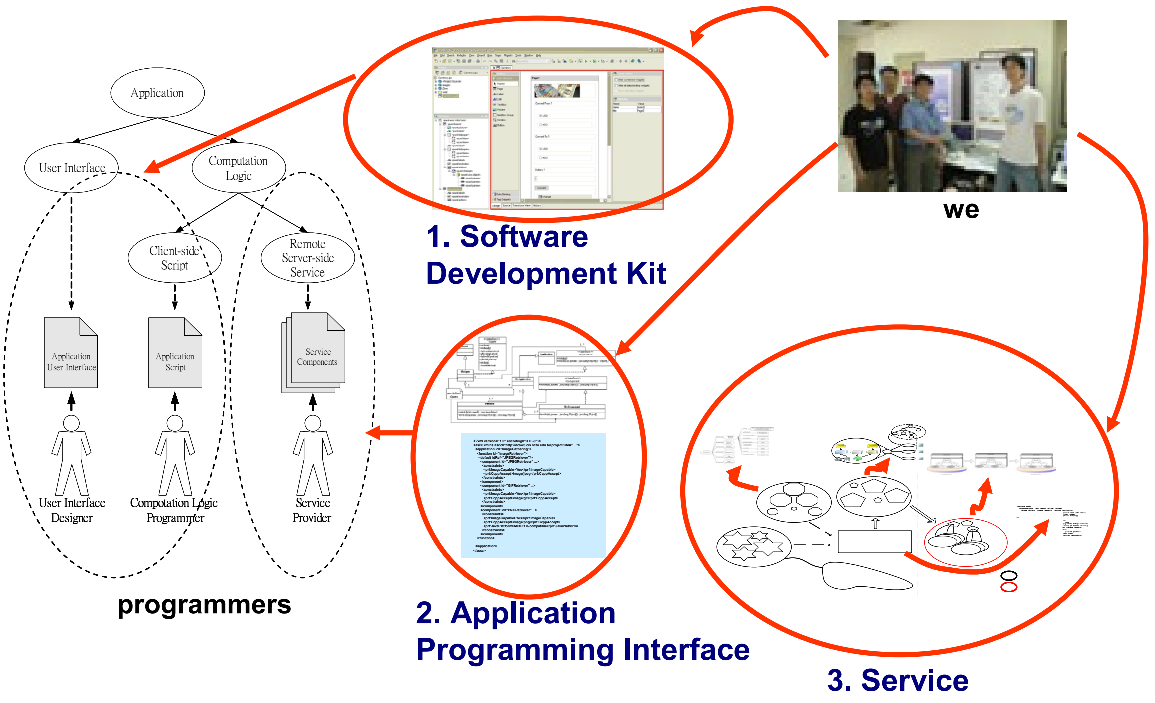


Fig. 7 The solution we propose

This paper is organized as follows. After an overall introduction to the context adaptation and transformation framework, we show context adaptation in Chapter 2. We will describe the essential aspects of PUML/PGML transformation in Chapter 3. Chapter 4 presents a system implementation and then explains the software development kit. In Chapter 5, we introduce the development steps on our system. Then, some related works are listed in Chapter 6. Finally, we discuss related work and conclusion.

Chapter 2
Context adaptation

In this chapter, we will explain the context awareness and adaptation process we attempt to use. We divide context information into to types. And to adapt applications, we structure an application, and use XML-based profile to describe it. Section 2.1 and 2.2 explain context awareness and the process for adaptation, respectively.

2.1 Context-awareness

To approach context adaptation, we collect two kinds of context information. The context information can be collected from many kinds of resources, such as http request with CC/PP and WAProf information, and the detected information from sensor devices, like, RFID. We divide them into two categories: Static and Dynamic, shown in Table 2.

Tab. 2 the types of context information

	Semantic	**Media**	**Format**
Static context information	device capability Description	File Access / Web Resources	W3C CC/PP WAP UAProf
Dynamic context information	Context information detected at runtime	RFID ZigBee	internal format

The use of the context profile to decide proper components in application adaptation is a critical issue. In this paper, we aim to develop an attribute-based algorithm that decides components appropriately by using CC/PP and WAP UAProf profiles.

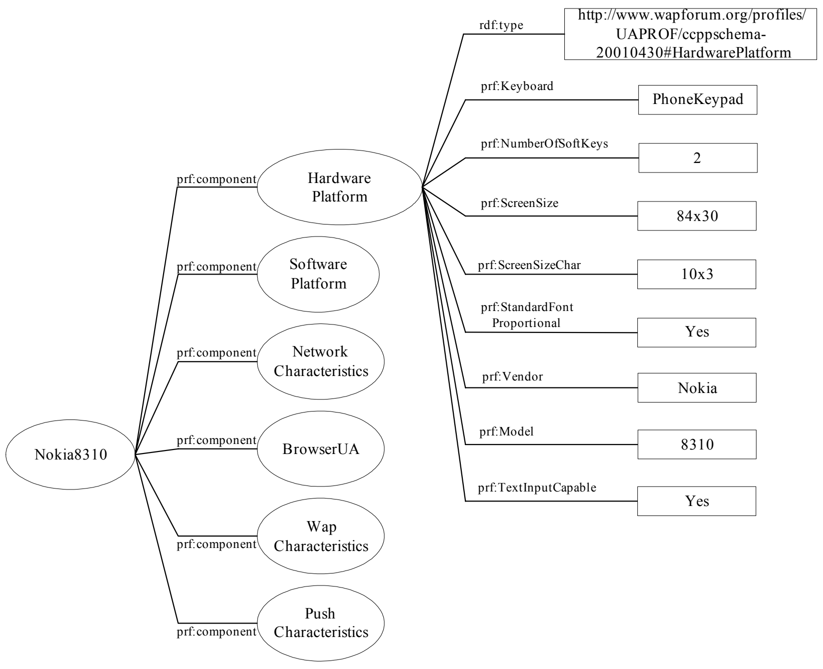


Fig. 8. A segmentation of Nokia 8310's WAP UAProf profile

Figure 8 illustrates a profile of WAP UAProf in RDF format. The Resource Description Framework (RDF) is a general-purpose language for representing information on the Web. This description covers six parts describing some characteristics of devices: Hardware Platform, Software Platform, Network Characteristics, BrowserUA, WAP Characteristics, and Push Characteristics. As in Figure 8, Hardware Platform specifies the hardware properties of devices. These properties include `prf:Keyboard`, `prf:NumberOfSoftKeys`, `prf:ScreenSize`, and `prf:ScreenSizeChar`, whose values can be `PhoneKeypad`, `2`, `84x30`, or `10x3` respectively. Listing 1 demonstrates the XML serialization form of the context profile in RDF.

Listing 1. The XML serialization form of the profile in Figure 8

```
<?xml version="1.0"?>
  <rdf:RDF ...>
    <rdf:Description rdf:ID="Nokia8310">
      <prf:component>
        <rdf:Description rdf:ID="HardwarePlatform">
          <rdf:type rdf:resource= "http://www.wapforum.org/profiles
           /UAPROF/cppschema-20010430#HardwarePlatform"/>
          <prf:Keyboard>PhoneKeypad</prf:Keyboard>
          <prf:NumberOfSoftKeys>2</prf:NumberOfSoftKeys>
          <prf:ScreenSize>84x30</prf:ScreenSize>
          <prf:ScreenSizeChar>10x3</prf:ScreenSizeChar>
          <prf:StandardFontProportional>
            Yes
          </prf:StandardFontProportional>
          <prf:Vendor>Nokia</prf:Vendor>
          <prf:Model>8310</prf:Model>
          <prf:TextInputCapable>Yes</prf:TextInputCapable>
        </rdf:Description>
      </prf:component>
      ...
    </rdf:Description>
  </rdf:RDF>
```

Figure 9 shows Gateway of Gateway (G^2), a system we implemented. The left-hand side is the RFID subsystem. There are three components containing tags, locators, and a reader. A user takes a tag. A locator is placed in a location, like in a room. When a user carrying tag enters a room, a locator will trigger this tag to send the user’s ID information. Then the reader will receive this information.

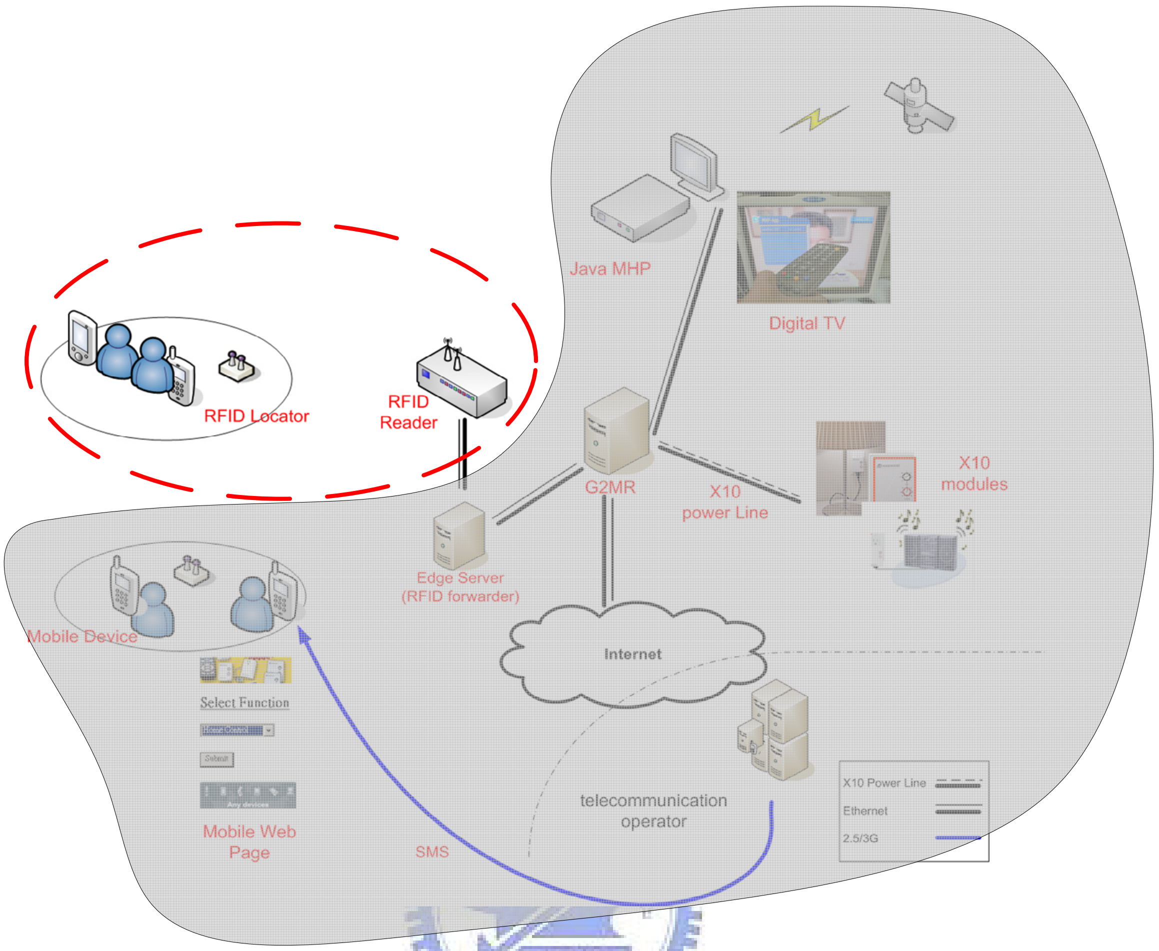


Fig. 9 RFID application

2.2 Context adaptation process

2.2.1 Application structure

We divide the application into two parts: the front-end module and back-end module. The front-end module contains two main constituents: an image display that can show images, and a requester that can send requests and receive the replied images. This module executes on the client device. The back-end module consists of four constituents: Image Transmitter, Cache, Image Retriever, Transcoder, and Data Access. In this module, Image Transmitter is responsible for receiving clients' requests and replying the required images. Image Retriever can obtain the requesting images from the biological multimedia information provider through Data Access. Next, it will pass the images to Transcoder for transforming images, such as image resizing, or to the cache for subsequent retrieving of images immediately. The following lists the possible procedures to retrieve the required images that the student wants to collect:

- 1→2→3→4→5→6→12→13: The procedure means that when Image Transmitter receives the user's request, the required images are obtained from the information provider.
- 1→2→7→8→9→12→13: This process refers to obtaining the required image from Cache.
- 1→2→7→8→9→10→11→12→13: This indicates that passing the images to Transcoder for resizing the images after Image Receiver gets the images in the second bullet, and then transmitting the result to the requester.

Figure 11 shows an application, which is structured hierarchically. This application is composed of more than one function that could be implemented by at

least one component. The application structure exhibited in Figure 10 can also be expressed in the form of the two-level hierarchy demonstrated in Figure 11. As we can see in Figure 11, each function links to candidate components. The function Image Retriever, for example, links to three components, JPEG Retriever, GIF Retriever, or PNG Retriever. This indicates that the function Image Retriever can be implemented by the three components.

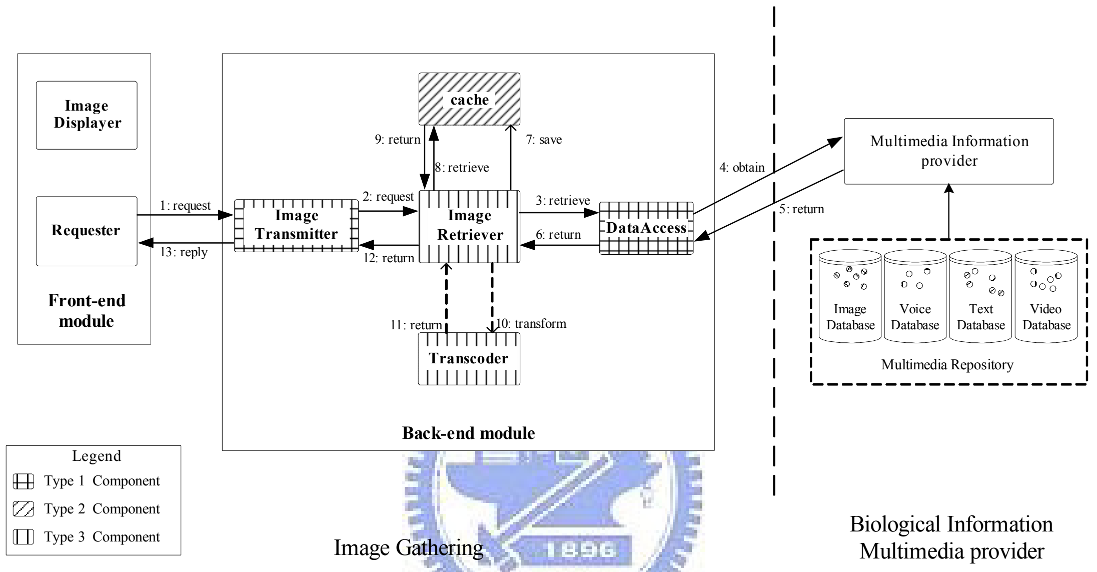


Fig. 10. The back-end module of the application Image Gathering

Components, in our system, are classified into three categories: Type1, Type2, and Type3. Type 1 components have the characteristics inclusive of *stateful*, *relative*, and *immoveable*. The stateful property means that the component records some particular data. For instance, Image Cache for the cache function belongs to this category. In contrast, specifying the stateful property No means that the components do not keep track of any particular data. If we declare a component as relative, it is associated with certain resources, and these components have database or TCP/IP connections. For example, the components of the function Data Access need to be

declared as this type, since it connects to the database of the multimedia provider over the networks. Another moveable property is used to modify the components that fail to be carried in agent migration.

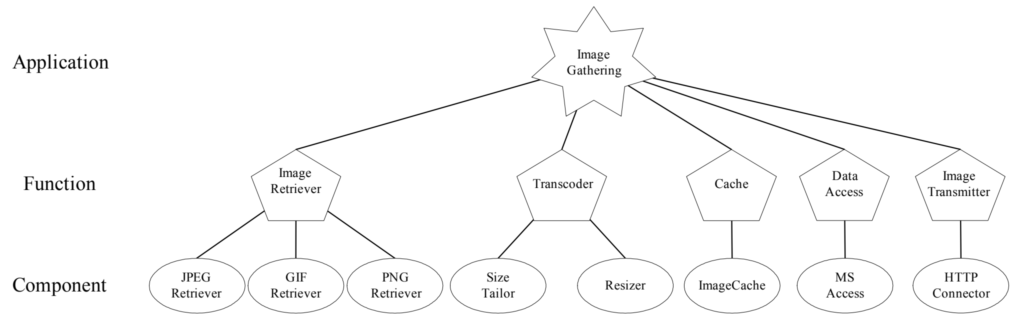


Fig. 11. The structure of the application ImageGathering

Type 1 components are the components connecting to certain resources. As demonstrated in Figure 10, candidate components of the function Data Access belong to this type. Type 2 components are those components which can be moved. The component Image Cache (in Listing 2) is declared as this type. Type 3 components are usually certain algorithms or pure computational logics, such as the XML transformation engine `javax.xml.transform.Transformer`. The following table arranges the three types of components.

Tab. 3 Three categories of components

Type	**Stateful/ Stateless**	**Relative/ Irrelative**	**Movable/ Immovable**	**Example**
Type 1	stateful	Relative	immoveable	Database Access
Type 2	stateful	Irrelative	moveable	Cache
Type 3	stateless	Irrelative	moveable	Transcoder

2.2.2 Attribute-based component decision algorithm

Take a profile Q for example. The profile Q includes a set of attributes, which can be expressed as $\{a_i \mid 1 \le i \le n\}$, where a_i and n denote an attribute and the total number of the attributes in the profile individually. Let $domain(a_i) = \{av_{i,k_i} \mid 1 \le i \le n \;\; and \;\; 1 \le k_i \le v_i\}$ indicate the domain of the attribute a_i, and $value[a_i]$ to be the value of the attribute, where v_i is the number of possible values of a_i. For instance, Listing xxx involves the attribute $TextInputCapable$, which has $value[TextInputCapable] = yes$ and $domain(TextInputCapable) = \{yes, no\}$.

An agent body contains a number of applications. An application comprises one or more functions fun_1, fun_2, ..., fun_m. Each of them can be implemented by at least one component, $comp_1^i$, $comp_2^i$, ..., $comp_{r_i}^i$, where $1 \le i \le n$ and r_i denotes the number of the user-defined components implementing fun_i. For example, fun_1 can be implemented by components $comp_1^1$ and $comp_2^1$ (illustrated in Figure 11). Each component $comp_y^x$ has a *constraint set* $CS_{x,y}$, which contains zero or more tuples $(a_i,\; av_{i,k_i})$, where $1 \le i \le n$ and $1 \le k_i \le |domain(a_i)|$, annotated under each component shown in Figure 12. We can accomplish the testing of a component to see if it can be chosen to implement its corresponding function by using this constraint set. For a component $comp_y^x$, if for the given profile Q, $comp_y^x$ can be chosen, it must be true that each attribute value av_{i,k_i} of $(a_i,\; av_{i,k_i})$ in its $CS_{x,y}$ is equal to the value of the same attribute a_i in Q. If so, we say that the component is satisfied. For example, assume that a certain profile and two components $comp_1^1$

and $comp_2^1$, and the function fun_1 are given. The component $comp_1^1$ has the constraint set $\{(ColorCapable,\ yes)\}$ and $comp_2^1$ has $\{(ColorCapable,\ no)\}$. Because the value of the same attribute $ColorCapable$ in this profile is yes, $comp_1^1$ is satisfied. $comp_1^1$ can be chosen to implement fun_1 accordingly. A constraint set, in implementation, can be established by a `<constraints>` element in the ASCC description. As in Listing 2, Lines 9-10 describe two elements, `<prf:ImageCapable>` and `<prf:CcppAccept>`. Therefore, the component $comp_1^1$ is declared suitable for processing JPEG files. As a result, the constraint set $CS_{1,1} = \{(ImageCapable,\ yes),\ (CcppAccept, image/jpeg)\}$ will be generated.

A component decision tree can be seen as a tree hierarchy. It comprises a number of attribute nodes, each of which has several branches linked to other attribute nodes as its child nodes. Let an_{i,d_i} indicate an attribute node, which is semantically equivalent to the attribute a_i with the same name in the given profile *Q*. Let av_{i,k_i} denote a branch of an attribute node an_{i,d_i}, where $1 \le k_i \le |domain(a_i)|$, d_i is between 1 and the component number at the same level in a tree, and $1 \le i \le n$.

Each attribute node an_{i,d_i} has a *linked component set* LC_{i,d_i} that includes the components associated via dotted lines in the component decision tree, illustrated in Figure 12. As in the figure, the linked component sets of the attribute nodes $an_{3,1}$ and $an_{3,2}$ are $LC_{3,1} = \{comp_{1,1},\ comp_{2,1}\}$ and $LC_{3,2} = \{comp_{2,2}\}$ respectively.

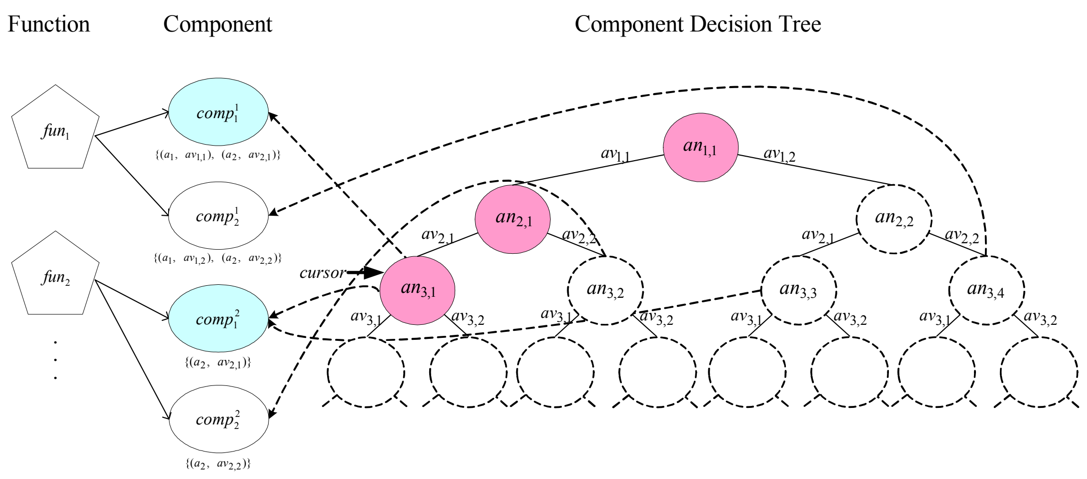


Fig. 12 A component decision tree and its linked components

To operate a component decision tree, there are a pointer *cursor*, and two operations, $NEXT(an_{i,d_i}, an_{i+1,d_{i+1}})$ capable of moving *cursor* from an attribute node an_{i,d_i} to its child node $an_{i+1,d_{i+1}}$, and $VISIT(an_{i+1,d_{i+1}})$ representing *cursor* visiting an attribute node $an_{i+1,d_{i+1}}$. Taking Figure 11 for example, the pointer *cursor* will point to the attribute $an_{3,1}$ since the operation $NEXT(an_{2,1}, an_{3,1})$ is applied. Thereupon $an_{3,1}$ is visited, denoted by $VISIT(an_{3,1})$. Furthermore, let *t* denote a traverse from the root to a certain leaf node. A traverse *t,* a sequence of $VISIT()$ and $NEXT()$, can be expressed as *SEQ(t)*=< $VISIT(an_{1,1})$, $NEXT(an_{1,1}, an_{2,d_2})$, $VISIT(an_{2,d_2})$, $NEXT(an_{2,d_2}, an_{3,d_3})$, ..., $NEXT(an_{i,d_i}, an_{i+1,d_{i+1}})$, ..., $VISIT(an_{n,d_n})$>. In Figure 12, for instance, a traverse *t* starts from the attribute node $an_{1,1}$ to the attribute node $an_{3,1}$. Thus, *SEQ(t)* is equal to < $VISIT(an_{1,1})$, $NEXT(an_{1,1}, an_{2,1})$, $VISIT(an_{2,1})$, $NEXT(an_{2,1}, an_{3,1})$, $VISIT(an_{3,1})$ >. Accordingly, while a traverse *t* is built, the linked component set LC_{i,d_i} of each

attribute node an_{i,d_i} visited can be united to establish a proper component set

$$P(t) = \bigcup_{\substack{\text{for each } i \text{ and } d_i, \\ \text{where } VISIT(an_{i,d_i}) \text{ in } SEQ(T)}} LC(an_{i,d_i}).$$

For example, suppose that there is a profile {*ImageCapable*, *CCPPAccept*, *JavaPlatform*, ...}, and their domains can be expressed *domain*(*ImageCapable*)={*yes*, *no*}, *domain*(*CCPPAccept*)={*yes*, *no*}, etc. Figure 13 demonstrates the structure of functions and components of an application. The function Image Retriever can be implemented by three components: JPEGRetriever, GIFRetriever, and PNGRetriever. The constraint set of the first component is $CS_{1,1}$ ={(*ColorCapable*, *yes*), (*CcppAccept*, *image/jpeg*)}, and that of the second component is $CS_{1,2}$={(*ColorCapable*, *yes*), (*CcppAccept*, *image/gif*)}. Moreover, in the component decision tree, each attribute node an_{i,d_i} has a number of branches and a linked component set LC_{i,d_i}. As in Figure 13, the attribute node *ImageCapable* has two branches, *yes* and *no*. The attribute node *CCPPAccept* has three branches encompassing *image/jpeg*, *image/gif*, and *image/png*. In addition, the attribute node $an_{3,1}$ is *JavaPlatform* whose linked component set is $LC_{3,1}$={*JPEGRetriever*}, and that of $an_{3,2}$ is $LC_{3,2}$ = {*ImageRetriever*}.

Let us assume that a traverse *t* is made by moving *cursor* from the root $an_{1,1}$ (*ImageCapable*) to the leaf node $an_{3,1}$ (*JavaPlatform*). As a result, the sequence *SEQ(t)=<VISIT(ImageCapable), NEXT(ImageCapable, CCPPAccept), VISIT(CCPPAccept), NEXT(CCPPAccept, JavaPlatform), VISIT(JavaPlatform)>* and the proper component set of *t*, *P(t)={JPEGRetriever}* are established.

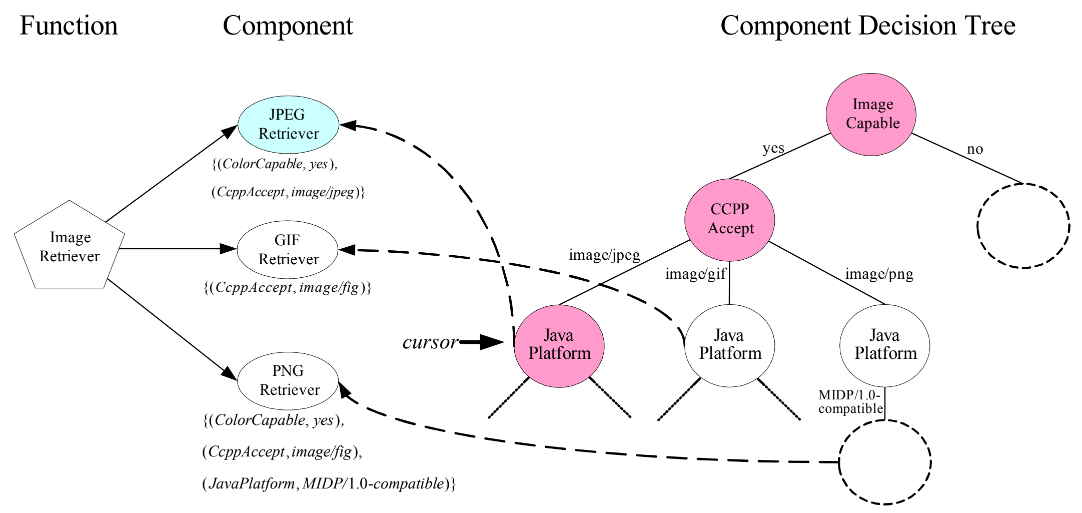


Fig. 13 An instance of a component decision tree on the right-hand side, and the associated components of the application ImageGathering on the left-hand side

The problem of how to decide a proper component to implement each function *f*, if given an application *p* and each function *f* of the application *p*?; or of how to adapt an application *p*, can be solved through the attribute-based component decision algorithm. This is because *SEQ(t)* and *P(t)* will be generated after traversing from the root to a leaf node. In *SEQ(t)*, $NEXT(an_{i,d_i}, an_{i+1,d_{i+1}})$ implies $VISIT(an_{i,d_i})$ and $value[an_{i,d_i}] = av_{i,d_i}$. Therefore, if a component $comp_y^x$ exists in *P(t)*, then $\forall i, k_i$ $value[an_{i,d_i}] = av_{i,d_i}$ = $value[a_i]$, where $an_{i,d_i} = a_i$ and a_i in $CS_{x,y}$. In other words, for a traverse *t* the proper component set *P(t)* contains the components, which are satisfied. Specifically speaking, given an application, if the suitable component exists for each function, this component can be chosen from *P(t)*. Moreover, if there are two or more suitable components at the same time, the last-examined component will be chosen as default. This algorithm solves the problem and eliminates the need for traversing a tree from the root to a leaf node. Once sufficient components exist in the proper component set *P(t),* traversing a component decision tree can terminate at

some internal attribute node which is not a leaf node.

In implementation, instead of realizing this algorithm by using the data structure tree, we realize this algorithm by means of a linking list. The reason for this is that using the tree as the data structure consumes more memory space to choose proper components. For each attribute node an_{i,d_i} at the same level of a component decision tree, the information recorded for the nodes seems different, except for the linked component set LC_{i,d_i}. However, they are essentially identical. Take the previous profile Q and the tree in Figure 12 for example. At level 3, $an_{3,1}$, $an_{3,2}$, $an_{3,3}$, and $an_{3,4}$ are semantically equivalent to the attribute a_3 in the profile Q. Therefore, to implement the concept tree, we use a linking list. In this way, for each level in a tree, attribute nodes an_{i,d_i}, for all d_i, where $1 \le i \le n$ and $1 \le d_i \le v_i$, are regarded as one node in a linking list. Figure 14 represents a linking list that starts from the root attribute node connecting to its child attribute node in the tree as the next node, which also links to its child node as the next node, and so on. This hierarchy of the linking list equals that of the component decision tree. In this list, an attribute node an_i has two links: one connects to a child attribute node an_{i+1}; the other binds its linked component set (a hash table in practice). In Figure 14, for example, the linking list, kept by a table index, starts from the attribute node an_1 to the attribute an_4, each of which binds a linked component set. For instance, the attribute node an_3 retains a link component set containing two components $comp_1^1$ and $comp_2^1$.

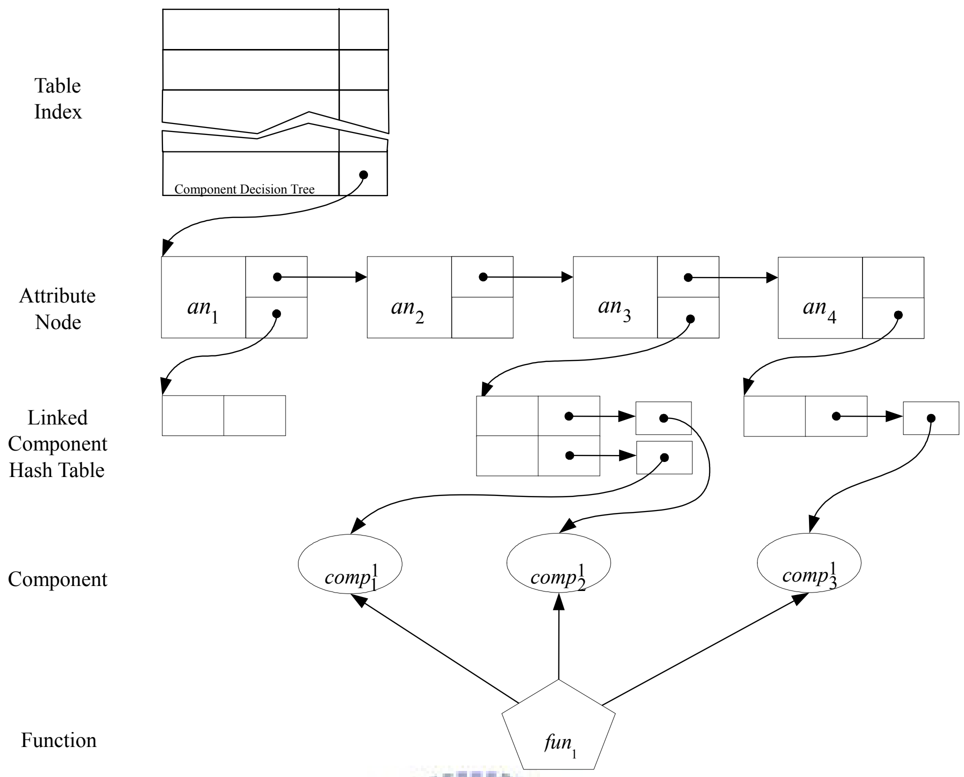


Fig. 14 The implementation (linking list) of a decision tree

Figure 15 illustrates the implementation of the component decision tree (Figure 12). Symmetrically, by traversing from the root node Image Capable to the node Java Platform, the proper component JPEG Retriever for the function Image Retriever can be decided.

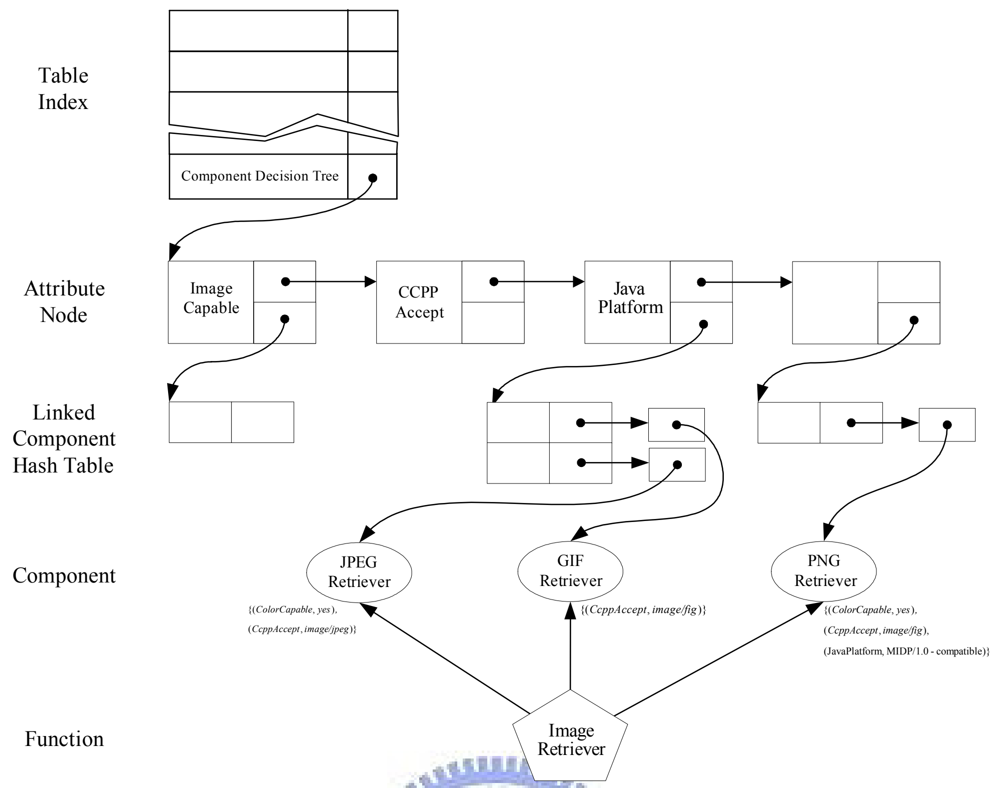


Table Index
Component Decision Tree
Attribute Node
Image Capable
CCPP Accept
Java Platform
Linked Component Hash Table
Component
JPEG Retriever
GIF Retriever
PNG Retriever
{(ColorCapable, yes), (CcppAccept, image/jpeg)}
{(CcppAccept, image/fig)}
{(ColorCapable, yes), (CcppAccept, image/fig), (JavaPlatform, MIDP/1.0 - compatible)}
Function
Image Retriever

Fig. 15. The linking list of the decision tree illustrated in Figure 13

In our system, we apply an attribute-based component decision algorithm to the application adaptation. Applications carried by the agent are adapted when the agent migrates to a new CAAS server. Implementing the component decision tree by a linking list simplifies the maintenance of attribute nodes. The space complexity is the sum of linked component hash tables $|\sum_{all\ d_i} LC_{i,d_i}|$ for all *i*, where $1 < i \le M$. It is less than $n_m * M$, where n_m is the number of attributes, and *M* is the size of the max linked component hash table. Besides, *M* is a constant. Therefore, the space complexity is $O(n_m)$.

In terms of time complexity, the time complexity is constant for the attribute-based algorithm as the processing time does not depend on the number of components. By contrast, we can consider a simple algorithm that decides proper components by examining each component. Thus, we inspect the constraint set

$CS_{x,y}$ for each component $comp_y^x$. This costs $O(n_c * n_m)$ worst-case time, where n_c denotes the total number of components of an application, and $n_m = max(|CS_{x,y}|)$ indicates the total number of attributes. The cost of the attribute-based algorithm is merely affected by the length of the linking list (the height of the component decision tree). In addition, the link can be built from the attributes in $CS_{x,y}$ for all components $comp_y^x$ in an application instead of generating from all attributes in a given profile. Therefore, its time complexity costs $n_m = max(|CS_{x,y}|)$. This means that the time complexity is dominated by the size of the max constraint set. As it can be seen, using the attribute-based algorithm to support decisions about component selection, facilitates programming of adaptive applications. It can support a large scale system with a large number of diverse implementations of particular functions.

2.2.3 Application Structure & Component Constraints

In order to enable this framework to be aware of the structures of applications, we define *Application Structure and Component Constraints* (ASCC), an application profile description. Listing 2 illustrates the ASCC profile of the application ImageGathering.

Listing 2. The ASCC profile to describe structure of ImageGathering

```
1 <?xml version="1.0"?>
2 <ascc xmlns:ascc=http://dcsw3.cis.nctu.edu.tw/project/CAAS ...>
3  <application id="ImageGathering">
4    <function id="ImageRetriever">
5      <default idref="JPEGRetriever"/>
6      <component id="JPEGRetriever" priority="51%"
7       stateful="No" relative="No" carried=”No”>
8        <constraints>
9          <prf:ImageCapable>Yes</prf:ImageCapable>
10         <prf:CcppAccept>image/jpeg</prf:CcppAccept>
11       </constraints>
12     </component>
13     <component id="GIFRetriever" priority="50%"
14      stateful="No" relative="No" carried=”No”>
15       <constraints>
16         <prf:ImageCapable>Yes</prf:ImageCapable>
17         <prf:CcppAccept>image/gif</prf:CcppAccept>
18       </constraints>
19      </component>
20      <component id="PNGRetriever" priority="50%"
21        stateless="No" relative="No" carried=”No”>
22          <constraints>
23            <prf:ImageCapable>Yes</prf:ImageCapable>
24            <prf:CcppAccept>image/png</prf:CcppAccept>
25            <prf:JavaPlatform>MIDP/1.0-compatible</prf:JavaPlatform>
26          </constraints>
27      </component>
28   </function>
29   <function id="Transcoder">
30     <default idref="SizeTailor"/>
31     <component id="SizeTailor" priority="50%"
32      stateful="No" relative="No" carried=”No”>
33     <component id="ColorTransformer" priority="50%"
34      stateful="No" relative="No" carried=”No”>
35   </function>
36   <function id="Cache">
37     <component id="ImageCache"
38      stateful="Yes" relative="No" carried=”Yes”>
39   </function>
40   <function id="DataAccess">
41     <component id="MSAccess" stateful="Yes" relative="Yes">
42   </function>
43   <function id="ImageTransmitter">
44     <component id="HTTPConnector"
46      stateful="Yes" relative="Yes"/>
47   </function>
48  </application>
49</ascc>
```

As we can see in Listing 2, the `<application>` element includes five `<function>` elements, which can describe the five functions. In each `<function>`, the candidate components can be specified. Lines 4-28, for instance,

declare that `<component id="JPEGRetriever" ...>`, `<component id="GIFRetriever" ...>`, and `<component id="PNGRetriever " ...>` can implement the Image Retriever function. In advance, within a `<component>` element, the properties, `stateful`, `relative`, and `carried`, can be used to set components stateful/stateless, relative/irrelative, and carried/un-carried respectively. The `priority` property concerns the priority of a component, one of which is chosen in each application adaptation. Furthermore, to set a component as a default component for a function, we can use the element `<default>`. If we want to set a component implementing the function which cannot be replaced with others, we can use the property "`unchanging='Yes'`".

Chapter 3 PUML/PGML Transformation

In this chapter, we will show the design of PUML and PGML, the intermediate languages for transformation. After the introduction, we will explain the transformation mechanism, and then the use of the mechanism context adaptation and PUML/PGML transformation.

3.1 Pervasive User Interface Markup Language (PUML)

3.1.1 Conceptual view

The principles of designing PUML including: (1) user interface abstracting; (2) intermediate language fashioning, and (3) OO (Object-oriented) [19] conceptualizing. To achieve the first, we analyze the characteristics of primary languages, including WML 1.1 [20], J2ME MIDP v1.0 [21], on the small and mobile devices. User interfaces of applications can be divided into two classes roughly. One is plentiful category; the other is fundamental category. User interfaces in the former has abundant widgets to display UI controls. User interfaces of the applications running on PCs, for instance, contains menu bar, tool bar widgets, etc. In addition, displaying

HTML documents on the desktop of PCs has a variety of modules, involving Frames module, Applet module, and etc. Respecting user interfaces on the screen of the small and mobile device, there are not a wide variety of widgets to render UI controls. It merely consists of basic presenting modules, such as Form module, Image module.

In this paper, we attempt designing form-based user interfaces to investigate device independence of the applications on the small and embedded devices. There are three primary reasons. First, the form module is the basic interaction component of user interfaces. Second, it is the module which is in the intersection of the UI components of J2ME and WML applications. It means that the widgets in the intersection module can be rendered both on the user interfaces of WML and J2ME applications. For example, bottom belongs to this kind of the widgets. Scroll bar, however, does not reside in the intersection, because it is incapable to be displayed on either one of them. Finally, to explore problems in approaching device independence via simple target languages, we can comprehend the use of markup languages to approach device independence easily. Moreover, we can gradually add other modules to enrich the PUML capabilities.

From the above considerations, we design PUML as a form-based user interface markup language. A user interface description of PUML comprises several containers. Each of the containers contains the basic widgets including label, text filed, single-choice listing, multi-choice listing, picture, and action. The following figure shows the conceptual view of PUML. The next section will detail the widgets and document structure of PUML, and a table shows the XML schema of PUML .

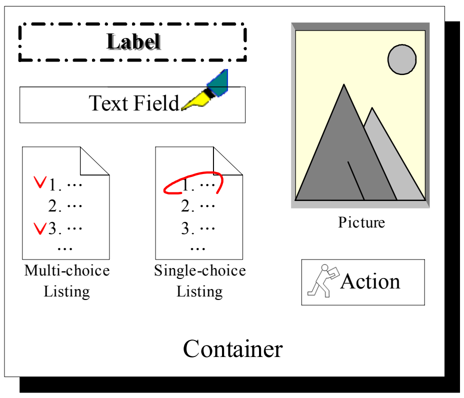


Fig. 16. The conceptual view of PUML

In terms of the second point, without doubt the language based on XML can serve as an intermediate and transformable language, explained in Section 1. In the third of the principles, we wish that writing PUML and PGML codes is in object-oriented manner. In such a way, programmers can write the PUML/PGML more intuitionally. Additionally, applications can be designed by applying the object-oriented analysis. For the reason, we design that to use a PGML description, e.g. a `.pgml` file, in a PGML document as an object used in Java [22]. The detail will be detailed in the following sections.

3.1.2 The language description

By exploiting XML, in PUML we define `<puml:label>`, `<puml:textnote>`, `<puml:listpaper>`, `<puml:picture>`, and `<puml:action>` to stand for label, text field, list, image, and action, respectively. These elements are the widgets which are involved in a `<puml:board>` container. Listing 3 illustrates the document structure of a user interface description using PUML. The DTD [23][24] of PUML can be referred in Appendix B.

The `<puml:user-interface>` element has three types of child elements, which are `<puml:logic-objects>`, `<puml:board>`, and `<puml:layout>`.

The `<puml:logic-objects>` element is used to declare the logic object, e.g. `.pgml` file, and the name used in a PUML document. A `<puml:board>` element can contain `<puml:textnote>`, `<puml:listpaper>`, `<puml:label>`, `<puml:picture>`, and `<puml:action>` elements; besides, each of the child elements can occur more than once. All of these elements can be rendered into UI controls capable of interacting with users. The detail of these elements will be explained in the next section.

A `<puml:layout>` element can specify the layout of the boards in a PUML document. On smart phones or some mobile devices, it has no effect, because their screens cannot display two or more boards meantime. However, if the description is migrated to be rendered on the screen of PCs, displaying of these boards is in disorder. Hence, we design the `<puml:layout>` element to state the arrangement of boards for extending the capability of PUML in the future. Other details concerning the elements will be explained in the next two sections.

Listing 3. An overview of the document structure of PUML:

```
<?xml version="1.0"?>

<puml:user-interface … >

 <puml:logic-objects>
   …
 </puml:logic-objects>

 <puml:board … >
   <puml:logic-objects>
     …
   </puml:logic-objects>

   <puml:textnote … />

   <puml:picture … />

   <puml:label … />

   <puml:listpaper … />

   <puml:action …>
     …
   </puml:action>
 </puml:board>

 <puml:board …>
  …
 </puml:board>

 <puml:layout … />
</puml:user-interface>
```

3.1.3 The elements of PUML

The top level elements are `<puml:logic-objects>`, `<puml:board>`, and `<puml:layout>`. A PUML document can has at most one `<puml:layout>` element. As mention in last section, it can be used to specify how to layout the board in that document. `<puml:logic-objects>` involves several `<puml:object>` elements, each of which declares a PGML document used in `<puml:board>` elements. Besides, the `<puml:logic-objects>` element can be directly applied into `<puml:board>` element. When a

<puml:logic-objects> element is used in a board, the visibility scope of the <puml:object> elements, declared within the <puml:logic-objects> element, is limited in this board. It means that the names of the <puml:object> elements, each of which stands for a PGML file, can be used in the board container only. <puml:board>, as mentioned above, is the basic container element, which includes the following child elements: <puml:picture>, <puml:label>, <puml:textnote>, <puml:listpaper>, <puml:action>, and <puml:logic-objects>. Moreover, it has the three attributes: name, title, and seqNo. The name attribute can be used to identify a board of the boards described in a PUML document. For the attribute, some value must be assigned by programmers. Concerning the meaning of the seqNo attribute, the seqNo attribute specify its priority. If the value of this attribute were 0, the board would be displayed first. If it were 1, the board, displayed on the screen, would follow the first one, and vise versa.

A <puml:picture> element, in a <puml:board> … <puml:board> block, can be used to show an image. It has four attributes: name, source, altText, and align attributes. A source attribute can be assigned to locate an image, which is specified to be exhibited. altText is the attribute which can be set a text value so as to be shown instead when the image cannot be displayed. The align attribute points the alignment of the image. Three attribute, left, center, and right, could be chosen to assign to this attribute.

A <puml:label> is similar to the label of the windows on the PCs. It can display a read-only text string on the screen. Its main attribute is showText, which can be assigned a string value to be displayed on the screen. We can use this element to show some title or prompt information. Oppositely, <puml:textnote> can get

a string from the user’s input. Namely, it allows users to input a certain value. Besides this attribute, the `type` attribute of the element can be assigned two values: `text` and `password`. If we set the attribute `text`, the value of the element would be rendered into the input value on the screen directly. If we set the attribute `password`, the value will be displayed with a character `*` to replace each character in the original string. It signifies that displaying of the input value is encrypted.

The `<puml:listpaper>` element serves as a choice group for picking a single item of a listing, or selecting a group of options among the items of the listing. The child content, within the element, includes at least one `<puml:item>` element as the optional items. To make the element be a multiple-choice or single-choice listing control, the `mode` attribute of the element can be set `multi` or `single`, respectively. Sharing the same function with the `name` attribute mentioned above, the `iname` attribute is assigned an ID for the element. Specifically, it emphasizes that it is a variable recording index values of the chosen items, instead of the values themselves. The `ivalue` attribute is the attribute which records indexes of the chosen `<puml:item>` elements in the `<puml:listpaper>` block. In `single` mode, the index value is an integer, counted from 1. The index value, differently, records a string value that could be `1`, `1;2`, `1;3`, `1;3;4`, and so on. As regards `<puml:item>` elements, each of them has a `showText` attribute displaying a prompt text to users, a `value` attribute recording the value of the item, and a `selected` attribute indicating whether it has been chosen.

`<puml:action>` elements can bind the events, which are triggered by the widgets of a PUML document, to the event-handling method of a PGML document. On the user interfaces of WAP-capable phones or J2ME smart-phones, it is rendered into a push item, which can be selected by users. For example, it could be a bottom if

it is displayed on the user interface of PCs. Child elements of the element could be `<puml:use-object>`, `<puml:change>`, and `<puml:nextboard>`. Using `<puml:use-object>` can declare that a name of an object, e.g. a PUML document, and the object owns a method to handle the events caused by the trigger of the action (shown as Line 3 in Listing 4). An object can be used in the board, if declared in this board or in the root element `<puml:user-interface>` via `<puml:use-object>` and `<puml:object>` elements. It means that the object, like a global variable, can be used in each of the `<puml:board>` elements within the `<puml:user-interface>` region.

To input parameters into a method, we can use `<puml:param>` elements as the child elements of the `<puml:use-object>`. Every `<puml:param>` element has a name and a value attribute; besides, the arrangement of the `<puml:param>` needs to conform to the order of the arguments of this method. Particularly, the usage of the attributes in a `<puml:param>` is either using only a `select` attribute, or using an attribute pair (`type` and `value`) alternatively. In the former, the `select` attribute can be assigned the value which refers to the `name` attribute of some widget element declared in the same board. It indicates that using this element as a variable; besides, the value attribute of this element can be retrieved at runtime. Alternatively, the latter can be used if programmers would like to input a value to the method directly. Line 5-8 and Line 10-12 in Listing 3 illustrate the two examples of using the `select` element and the `type` and `value` pair, respectively. In addition, the returned value from the method can update the attribute value of some widget which is specified by means of the `<puml:change>` element, shown in Line 14-19. Respecting the `<puml:nextboard>` element (Line 21), an name of some board can be assigned to the `goto` attribute of the element. It can change the screen to display the board

specified, when its parent element `<puml:action>` is triggered.

Listing 4. Examples of <puml:use-object>, <puml:param>, and <puml:change>

```
<puml:action name="action" showText="actionDemo">

  <puml:use-object name="object1" method="getRandNum" />

  <puml:use-object name="object1" method="getMax">
    <puml:param select="note1" />
    <puml:param select="note2" />
  </puml:use-object>

  <puml:use-object name="object1" method="getAbsVal">
    <puml:param type="int" value="-1" />
  </puml:use-object>

  <puml:change container="board1" component="note2" update="value">
    <puml:use-object name="object1" method="getMax">
      <puml:param select="note1" />
      <puml:param select="note2" />
    </puml:use-object>
  </puml:change>

  <puml:nextboard goto="board2" />

</puml:action>
```

3.2 Pervasive LoGic Markup Language (PGML)

3.2.1 Design Principle

In [15], we mentioned the markup language of the event-handling logic. Currently, we have revised the language and rename it Pervasive loGic Markup Language (PGML). We endeavor to enable the language to support a wide range of devices to approach device independence. Initially, we concentrate on applying the language to the small and mobile devices. Like PUML, the design principles of the language we consider are: (1) computation generalizing, (2) intermediate language fashioning, and (3) OO conceptualizing.

The first means that, logic languages, such as C, Java, or WML Script, have the primary statements to declare variables and function blocks, and flow-control and condition-control mechanisms to complete basic computation. Therefore, we abstract the primary expressions and statements to define PGML. The second and third points are explained in Section 1 and 2.1. From the three points, we make PGML own the following capabilities:

- The object-oriented concept
- Local and global variables declaration
- Mathematical, logical, and boolean expressions
- Flow-control and condition-control statements
- Method declarations.

3.2.2 The design of PGML

Peripherally the features of PGML and PUML are different, but essentially the two languages have the same intention. PGML aims to describe the computational logic in the XML format. The computational logic is a section of an application, containing mathematical, comparison computations, and method invocations, etc. Figure 17 demonstrates a PGML object in a tree view of the XML Schema [25]. An object, e.g. a PGML file, is composed of a variable declaration `<declaration>` and a least one method `<method>`. Listing 5 shows an overview of the PGML document structure.

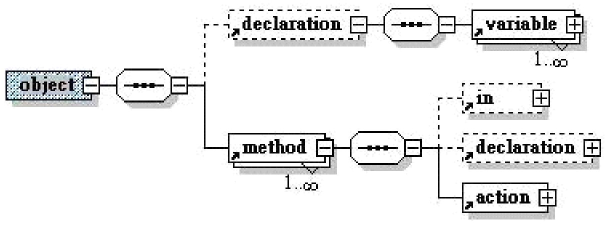


Fig. 17. A tree view of the XML Schema of PGML

Listing 5. The document structure of PGML:

```
<?xml version="1.0"?>
<pgml:object … >

  <pgml:declaration>
    …
   <variable …>
    …
  </pgml:declaration>

  <pgml:method … >

    <pgml:in> … </pgml:in>

    <pgml:declaration> … </pgml:declaration>

    <pgml:action … >
      …
      <pgml:if> … </pgml:if>
      <for> … </for>
      <assign … > … </assign>
      …
      <pgml:return … />
    </pgml:action>

  </pgml:method>

</pgml:object>
```

To explain expressing computational logic in XML in advance, we take four simple examples of `<add>`, shown in the following listing. Line 2-5, Line 7-10, and Line 12-15 denotes that `a+b`, `5+6`, and `s=5+6`, respectively. In detail, Figure 18 presents that an `<add>` contains two child elements as its operands. One of the two elements could be one in the elements of the `MathExpression` Group, or in those of the `OperandGroup` group, such as the `<operand>` element. One of them, for example, can be a `<add>` element (Line

17-23), and consequently the code of PGML will be transformed into `s=(3+4)+5`. Other cases of using PGML and converting PGML expressions into the Java syntax are illustrated in Appendix A.

Listing 6. The document structure of PGML:

```
1
2   <pgml:add>
3     <pgml:operand select="a"/>
4     <pgml:operand select="b"/>
5   </pgml:add>
6
7   <pgml:add>
8     <pgml:operand value="5" type="int" />
9     <pgml:operand value="6" type="int"/>
10  </pgml:add>
11
12  <pgml:add result="s" >
13    <pgml:operand value="5" type="int" />
14    <pgml:operand value="6" type="int"/>
15  </pgml:add>
16
17  <pgml:add result="s" >
18    <pgml:add>
19      <pgml:operand value="3" type="int" />
20      <pgml:operand value="4" type="int"/>
21    </pgml:add>
22    <pgml:operand value="5" type="int"/>
23  </pgml:add>
```

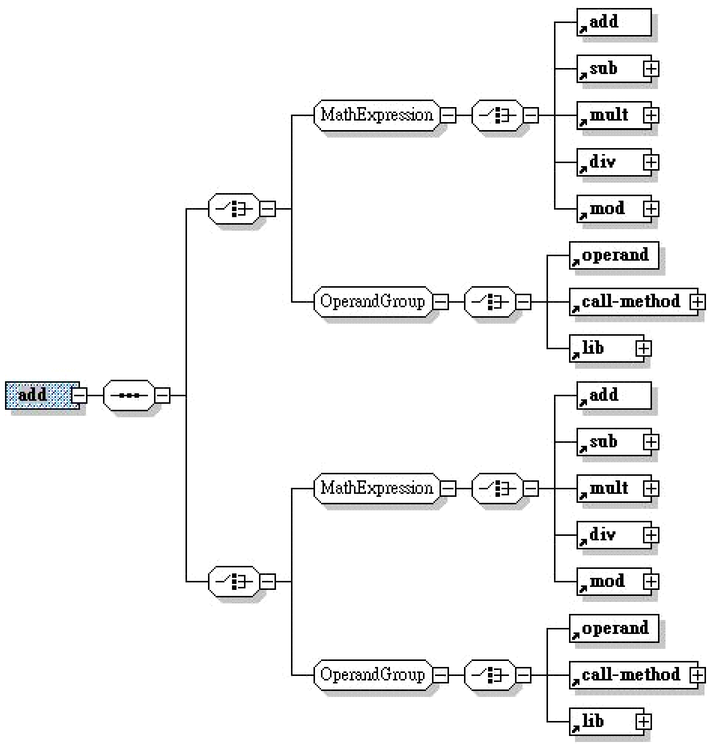


Fig. 18 the XML Schema description of the `<add>` element

By composing these statements in the functions of a PGML document, each of these can be specified to deal with an event triggered by a widget of the user interface. A statement is composed of several expressions, each of which is a series of variables, operators, and method calls. Through PGML, a logic programmer can write compound expressions by combining expressions to construct the logic section of an application in the XML format. For example, a programmer layouts a button on the user interface of an application, and he can write the `addTwoNum()` method in the PGML document to handle the button pressed, shown as listing 7.

Programmers can use PGML elements to write the computational logic. For instance, the `<pgml:method>` element expresses a method declaration block. The `<pgml:in>` element contains some child elements, which are the arguments needed passing into this method. In PGML, there is a element `<pgml:init>` similar to the `<pgml:in>` element. Differently, it is used to declare and initiate the local variables

in a flow-control element `<pgml:for>…</pgml:for>`, or a method declaration block `<pgml:method>…</pgml:method>`. The following listing demonstrates a PGML example, and Listing 8 shows the Java source program transformed from the PGML code. The detail related to the transformation mechanism will be explained in the next section.

Listing 7. A method declaration in PGML:

```
<?xml version="1.0"?>
<pgml:object name="addTwoNum" version="1.0"
 xmlns:pgml="http://dcsw3.cis.nctu.edu.tw/Project/
 pervasive/PGML/" >

 <pgml:method name="sum"  visibility="public" return-type="int">

   <pgml:in>
     <pgml:variable name="num1" type="int" />
     <pgml:variable name="num2" type="int" />
   </pgml:in>

   <pgml:action>
     <pgml:return>
       <pgml:add>
         <pgml:operand select="num1" />
         <pgml:operand select="num2" />
       </pgml:add>
     </pgml:return>
   </pgml:action>

 </pgml:method>

</pgml:object>
```

Listing 8. The Java code transformed from the PGML document in Listing 7:

```java
package SumExample;
import SumExample.*;
import java.lang.Integer;
import java.lang.String;

public class addTwoNum {
  public addTwoNum(){ }
  public int sum( String  sys_num1, String  sys_num2 ){
    int  num1= Integer.parseInt( sys_num1);
    int  num2= Integer.parseInt( sys_num2);
    return ( num1 + num2 );
  }
}
```

3.3 Leveraging XSLT/XPath Transformation

3.3.1 An overview of language transformation mechanism

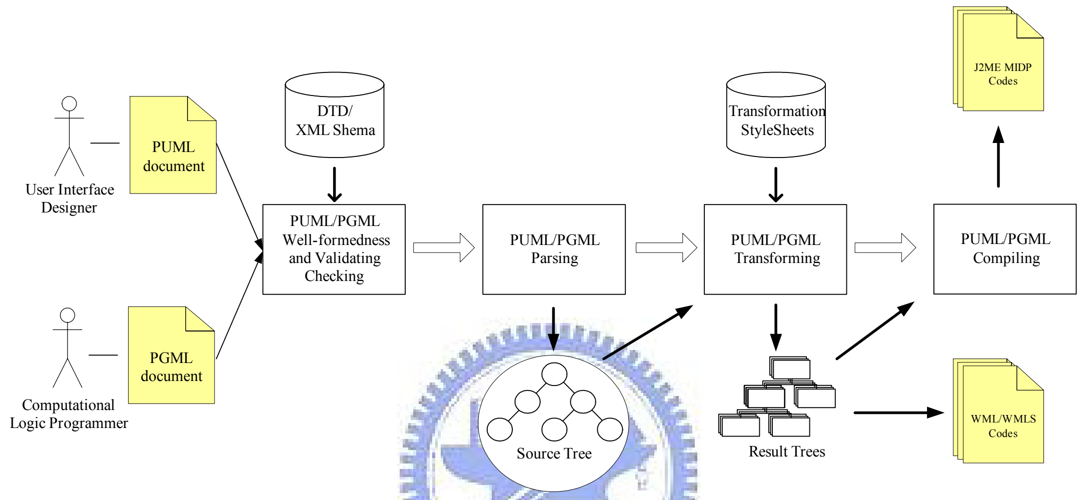


Fig. 19. The processing flow of PUML/PGML documents

In [18], we attempted the context-aware adaptation first time by using CC/PP and XSLT. Then we have refined that and designed PUML and PGML for context aware adaptation and transformation. Figure 19 is the overview of transforming the PUML and PGML documents into the J2ME and WML languages. To transform PUML/PGML, we provide a basic development toolkit involving *PUML/PGML checker*, *PUML/PGML parser*, *PUML/PGML transformer*, and *MExE Language Compiler* to complete the transformation process. The process contains the following four main steps:

PUML/PGML well-formedness checking and validating: When finishing writing a PUML/PGML document, a programmer can use the development toolkit to check the well-formedness and validity for the PUML/PGML document.

PUML/PGML parsing: With no error found in the document, the document will be parses into a DOM tree.

PUML/PGML transforming: The PUML/PGML transformer, deriving form the transformation engine Xalan-Java 2.5.1 [63], is capable of transforming the tree into the result trees through each transformation stylesheet for the target languages. Each of stream results (`javax.xml.transform. stream.StreamResult`) formed form a result tree is generated. Besides, the stream results are serialized into source programs. The transformation procedure will be explained in the next section.

PUML/PGML compiling: MExE Language Compiler (mexe-compiler) can compile each of the source programs generated into its specific executable code. For example, for the J2ME and PJava platforms, a Java source program (a `.java` file) generated is compiled into Java bytecodes (`.class` files). Nonetheless, not all the generated source programs need to be compiled. `.wml` (WML) and `.wmls` (WML Script) codes need no compilation, since they can be interpreted by user agents of client devices. From this point, if the source programs of Java were generated, they would be compiled into executable codes by the mexe-compiler. Otherwise, when the programs of WML and WML Script are generated, they are not compiled.

3.3.2 XSLT transformation procedure

This section will explain how to achieve transforming PUML and PGML documents through XSLT/XPath [63]. To explain the mechanism, we use Listing 5 as the input PGML document and the PGMLtoJ2MEMIDP.xsl styleschеet capable of transforming a PGML document into a J2ME document. Retrieving from the stylesheet, four templates, `<xsl:template match="/">`, `<xsl:template match="pgml:object">`, `<xsl:template`

match="pgml:method">, <xsl:template match="pgml:in">, and <xsl:template match="pgml:variable"> are shown in Listing 7.

The inputted PGML document is parsed into a source tree. Then, the transformer would traverse the nodes of the input source tree from the root node to the leaves. At the beginning, the first template will be fired once the root node is encountered. A J2ME code, which is described within the <xsl:template match="/"> … </xsl:template>, is generated thereupon. In XSLT, <xsl:value-of … /> can be used to retrieve the name of the node which is visited, and the values of the attributes of this node; moreover, the <xsl:apply-templates … /> instruction is used to process all of the children of the current visited node, and to fire the matching templates successively.

In this way, the code generated is illustrated in Step 1 of Table 1. The other four steps demonstrate generating the J2ME code from the input PGML document by applying the last four templates, exhibited in Listing 7. Step 2 will occur when the transformer visits the root element <puml:object>. Step 3 shows that the <puml:method> node is visited, and next the a code of the sum() method is outputted via the <xsl:template match="pgml:method"> template (Line 21-30 in Listing 7). In the template, <xsl:value-of select="@name"/> is used to obtain the value of the name of <pgml:method name="sum" … > (Line 6 in Listing 5). Afterwards, <xsl:template match="pgml:variable"> will be applied to generate the code of the argument of the method. After the transformer has completed traversing the input PGML document, the code which is shown in Listing 6 will be generated eventually.

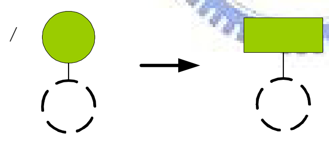
Listing 9. Four templates in the PGML-to-J2MEMIDP transformation stylesheet

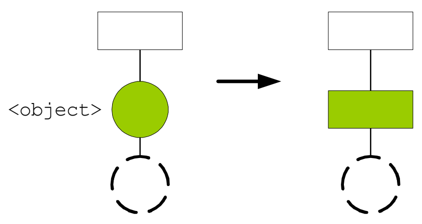
```
<xsl:template match="/">
  <xsl:text>package </xsl:text>
  <xsl:value-of select="$pgml:packageName"/>
  <xsl:text>;</xsl:text>
  import <xsl:value-of select="$pgml:packageName"/>.*;
  import java.lang.Integer;
  import java.lang.String;
  <xsl:apply-templates select="pgml:object"/>
</xsl:template>

<xsl:template match="pgml:object">
  public class <xsl:value-of select="@name"/> {
    <xsl:apply-templates select="pgml:declaration" />
    public <xsl:value-of select="@name" />(){}
    <xsl:for-each select="pgml:method">
      <xsl:apply-templates select="." />
    </xsl:for-each>
  }
</xsl:template>

<xsl:template match="pgml:method">
  <xsl:value-of select="@visibility"/>
  <xsl:value-of select="@return-type"/>
  <xsl:value-of select="@name"/>( <xsl:apply-templates
    select="pgml:in"/> ){
  <xsl:apply-templates select="pgml:declaration"/>
  <xsl:call-template name="declarationTempVar" />
  <xsl:apply-templates select="pgml:action"/>
}
</xsl:template>

<xsl:template match="pgml:in">
  <xsl:for-each select="pgml:variable">
    <xsl:apply-templates select="."/>
    <xsl:if test="not(position()=last())">
      <xsl:text>, </xsl:text>
    </xsl:if>
  </xsl:for-each>
</xsl:template>

<xsl:template match="pgml:variable">

  <xsl:choose>

    <xsl:when test="node()">
      <xsl:value-of select="@type"/>
      <xsl:value-of select="@name"/>
      <xsl:text>=</xsl:text>
      <xsl:apply-templates/>
    </xsl:when>

```

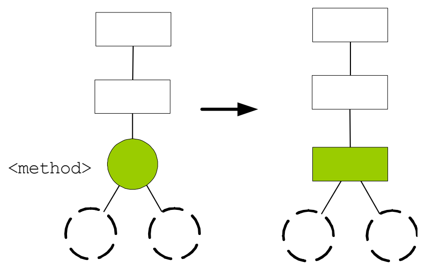
```
  <xsl:otherwise>
   <xsl:if test="not(name(..)='pgml:in')">
    <xsl:value-of select="@type"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="@name"/>
    <xsl:if test="@value">
     <xsl:text>=</xsl:text>
     <xsl:call-template name="process-type-of-value"/>
    </xsl:if>
   </xsl:if>
   <xsl:if test="name(..)='pgml:in'">
    <xsl:text>String  sys_</xsl:text>
    <xsl:value-of select="@name"/>
   </xsl:if>
  </xsl:otherwise>

 </xsl:choose>

 <xsl:call-template name="test-statement-end" />

</xsl:template>
…
```

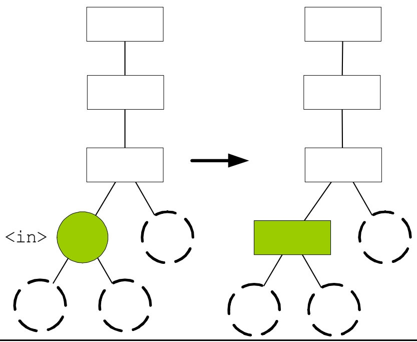
Tab. 4 The result of the first five steps of transforming the PGML document into the J2ME MIDP code

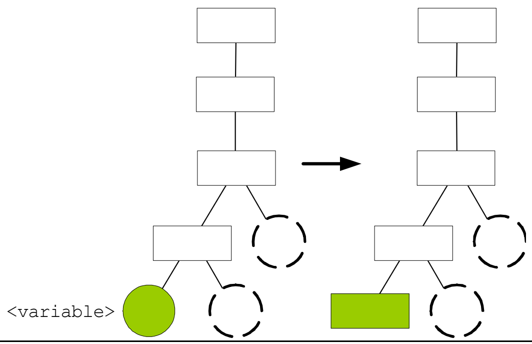
No. of Step	Matching template	J2ME MIDP code generated

Step 1		package SumExample;
		
		import SumExample.*;
		import java.lang.Integer;
		import java.lang.String;

Step 2		package SumExample;
		
		import SumExample.*;
		import java.lang.Integer;
		import java.lang.String;
		
		public class addTwoNum {
		  public addTwoNum(){}
		}

Step 3		import SumExample.*;
		import java.lang.Integer;
		import java.lang.String;
		
		public class addTwoNum {
		  public addTwoNum(){ }
		  public int sum(){ }
		}

Step 4		`import SumExample.*;`
		`import`
		`java.lang.Integer;`
		`import java.lang.String;`
		
		`public class addTwoNum {`
		`  public addTwoNum(){ }`
		`  public int sum( ,){ }`
		`}`

Step 5		`import SumExample.*;`
		`import`
		`java.lang.Integer;`
		`import java.lang.String;`
		
		`public class addTwoNum {`
		`  public addTwoNum(){ }`
		`  public int sum(`
		`   String  sys_num1, ){ }`
		`}`

…	…	…

Similarly, we apply the XSLT/XPath technique to transform PUML documents. We define two transformation stylesheets, PUMLtoWML.xsl for transforming PUML documents into WML codes, and PUMLtoJ2MEMIDP.xsl for transforming the same inputs into J2ME MIDP codes. Table 2 lists the mappings of the elements of the user interface from the PUML elements to its related J2ME MIDP expressions, which is used in PUMLtoJ2MEMIDP.xsl. Table 3 indicates the same PUML elements and the associated WML tags which the PUML elements are converted into.

Tab. 5. The mappings from PUML tags into MIDP expressions in the PUML-to-J2ME transformation stylesheet

	PUML Tag	**WML**
Root element	`<user-interface>`	`<wml>`
Container	`<board>`	`<card …> … </card>`
Text String	`<label>`	`<label … />`
Text Field	`<textnote>`	`<input … />`
Selection List	`<listapper>`	`<select …> … </select>`
List Item	`<item>`	`<option … />`
Image Display	`<picture>`	`<img … />`

Action Trigger	`<action>`	`<do>`
		`  <go … />`
		`</do>`

Tab. 6. The mappings from PUML tags into WML expressions in the PUML-to-WML transformation stylesheet

	PUML Tag	**MIDP Expression**
Root element	`<user-interface>`	`javax.microedition.lcdui.MIDlet`
Container	`<board>`	`javax.microedition.lcdui.Form`
Text String	`<label>`	`javax.microedition.lcdui.StringItem`
Text Field	`<textnote>`	`javax.microedition.lcdui.TextField`

Selection List	`<listapper>`
	`  <item>`
	`  …`
	`</listpaper>`
		`javax.microedition.lcdui.ChoiceGroup`

Image Display	`<picture>`	`javax.microedition.lcdui.ImageItem`
Action Trigger	`<action>`	`javax.microedition.lcdui.Command`

3.4 The use of combination of context adaptation and XSLT transformation

Figure 20 illustrates our capability of combining the transformation and adaptation schemes. This figure shows that the function Page Generate can be implemented by three components: XHTML MP Generator, WML Generator, and CHTML Generator. Each component is implemented by using XML transformation engine. Through the use of different stylesheets, these generators can generate the target languages. This method can be used in an application serving users the displayed pages in those different target languages.

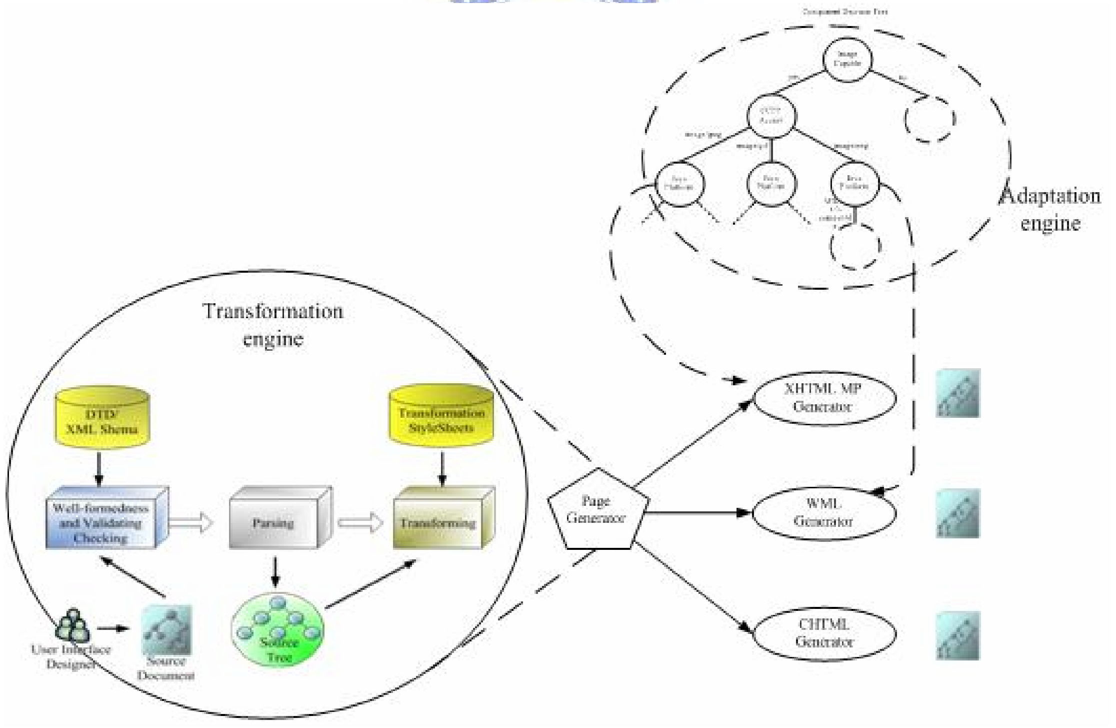


Fig. 20 Integrating transformation and adaptation

Chapter 4
System implementation

This chapter will introduce three system implementations. Section 4.1 explains Context Aware Adaptation Service (CAAS) [16]. Section 4.2 and 4.3 introduce Ubi-Adapting and G^2 respectively.

4.1 Context-Aware Adaptation Service (CAAS)

4.1.1 Example scenario

The last section has mentioned the main objectives and our focuses. In this section, we explain the architecture and components of our framework. Figure 21 illustrates a scenario of an application, which can be developed by our framework. Assume that in a campus there is a wired Ethernet and IEEE 802.11 built; users can surf the Internet via the networks. In the campus, a student, who wants to gather the butterfly pictures, can collect the information from the biological multimedia information provider via his personal computer when he is in his laboratory. Then, when he goes out for a meeting (the arrow indicates the direction of his moving), he can use a PDA to continue collecting the images. His work also can be kept on by borrowing a notebook from his colleague, after arriving in the meeting room.

This scenario involves two critical techniques. First, the image can be resized suitably according to the context of the device. Second, the collection status can be kept on executing without interruption after changing his device, and even if he is

moving. In order to approach it, we attempt to design our framework can provide the following functionalities, described below:

- It can divide a program of the application into two modules: one is back-end module running at the server side for retrieving the images and transmitting the images to users' devices; the other is front-end modules executed on users' devices for representing the gathered images, shown in Figure 18.
- Numerous computing transformation and adaptation algorithms needing heavy computational resources will be enveloped in the back module to execute at server side instead of running the whole program on the devices. Thus, the restriction of application development by the limitation of resources on devices can be reduced.
- The system can change the transformation and adaptation component of the application to others appropriately depending on what devices users use.
- The back-end module can move with the user.

4.1.2 System overview

Figure 21 exhibits the infrastructure of our system. There are three main components: *context-aware adaptation*, *repository service*, and *client*. The client devices can be the mobile and embedded appliances or stationary computers, such as PCs, PDAs, laptops, smart phones, Java phones, etc. A front-end module for each user can be executed on its owner's device. Underlying the devices, the access networks include wireless IEEE 802.11x networks, 2.5/3G telecommunication networks. At the server side, there is a context-aware adaptation server (CAAS server) in each local area network (LAN). In our design we do not assert that each LAN must have a CAAS server, but if we assume there is a server in a LAN. The application can benefit by that when a user

enters this local area network, his personal agent can carry back-end modules and migrate to a nearby server close to him. Besides CAAS servers, there is a repository service, which supplies CAAS servers with the registered users' information and structure descriptions of deployed applications.

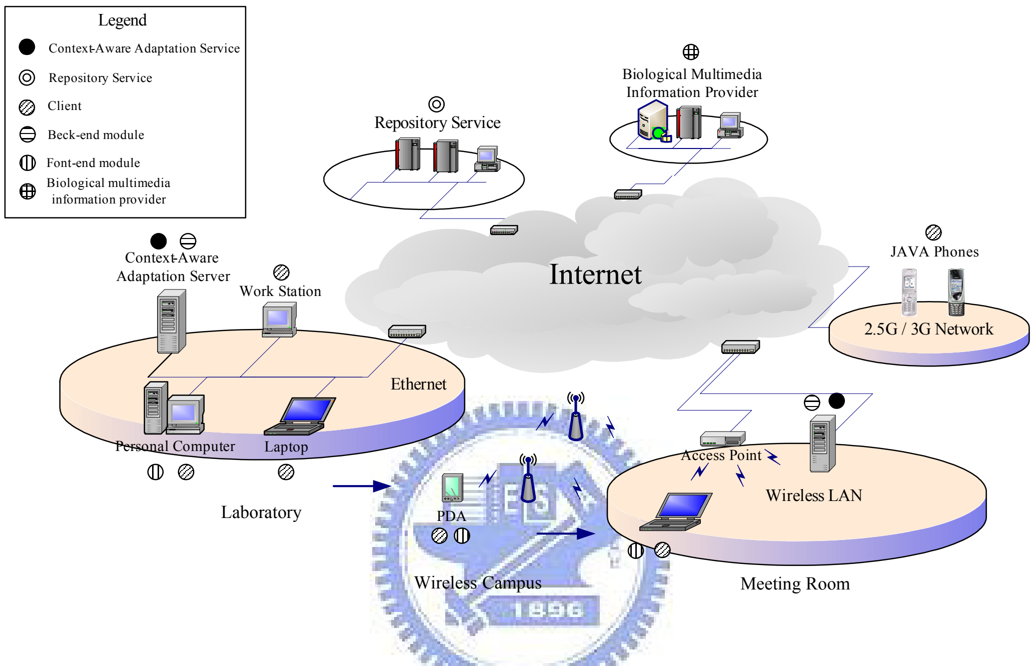


Fig. 21. An overview of the system infrastructure

4.1.3 System architecture

Figure 21 demonstrates the infrastructure of our system. Internally, there are three primary constituents, client, context-aware service, and repository service, which correspond to *Client Tier*, *Context-aware Service Tier*, and *Repository Service Tier* respectively, presented in Figure 22.

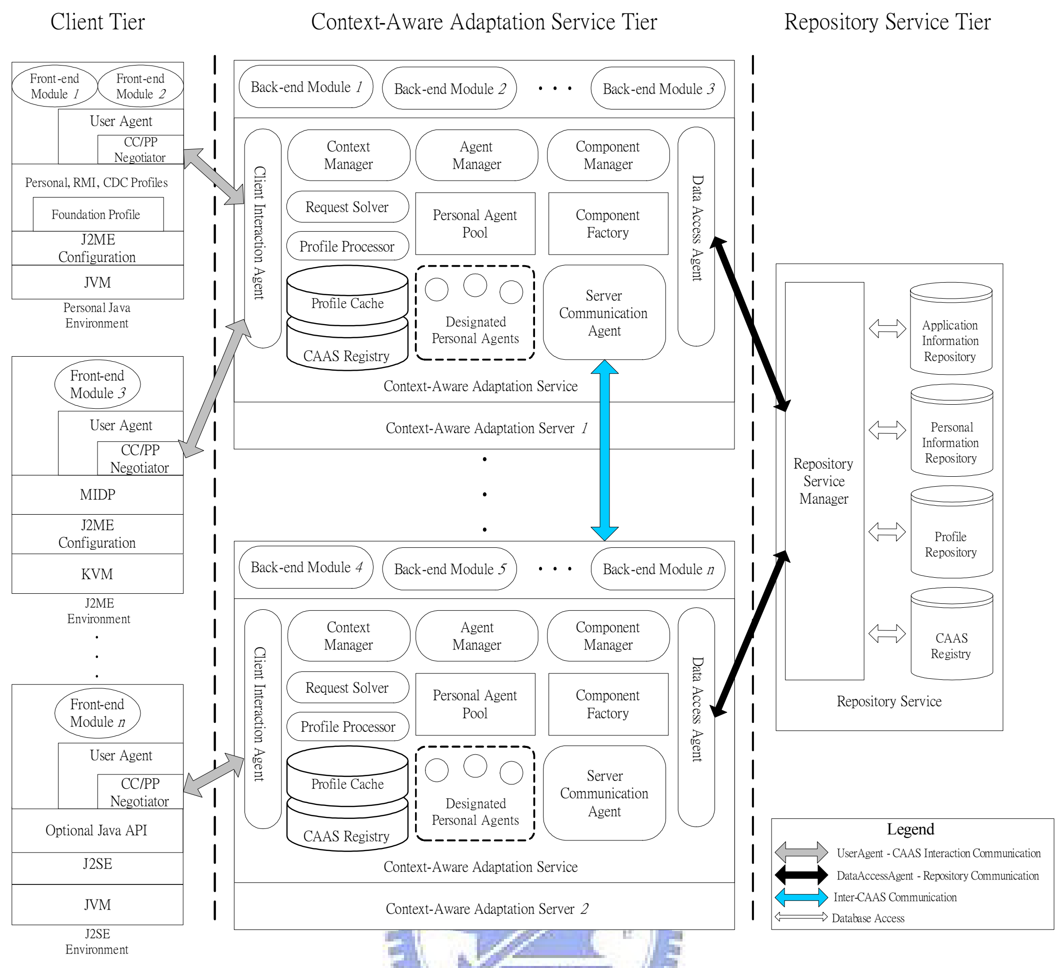


Fig. 22. The inner architecture of the context-aware adaptation framework

Client tier:

This tier contains various mobile, handheld, or stationary computing appliances. As in Figure 22, the devices cover J2ME-capable phones, PJava-capable PDAs, and J2SE-runable laptops or personal computers. PJava, J2ME [8][9], and J2SE are the Java virtual machines for the different kinds of operating systems and hardware environments. Each of them has some particular configuration profiles. In J2ME, KVM, J2ME Configuration, and MIDP are involved. In addition to the functions of J2ME, PJava includes Configuration, Foundation Profile, Personal RMI, CDC Profiles yet, whereas the J2SE environment comprises JVM and optional Java API, furthermore.

A device can send a registering message with its context profile to inform a CAAS server of its capabilities, and a front-end module on the device can invoke the methods of their back-end modules via remote dynamic invocation (explained in Section 6). To implement these mechanisms, two components of our system serve on the client devices. One is *User Agent* that provides the `UserAgent` API to invoke the methods of back-end modules. The other is *CC/PP Negotiator*, which can transmit its device's CC/PP profile embedded in a registering message to a CAAS server, when the user's device initially connects to this server. This messaging is based on the CC/PP content negotiation protocol [26[27][29]. Intrinsically, messages of the negotiation protocol can be sent by wrapping them in HTTP request/reply messages. Listed below is RDF/XML [28][30][31] serialization of a context profile contained in the messages, and illustrates that the number of pixel is 16 in the hardware component of a user device.

Listing 10. A CC/PP profile

```
GET /ccpp/html/ HTTP/1.1
...
<?xml version="1.0"?>
     <rdf:RDF ...>
     <rdf:Description rdf:ID="MyDeviceProfile">
        <prf:component>
           <rdf:Description rdf:ID="HardwarePlatform">
              <rdf:type rdf:resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-
                20010426#HardwarePlatform"/>
              <prf:BitsPerPixel>16</prf:BitsPerPixel>
           </rdf:Description>
        </prf:component>
     </rdf:Description>
  </rdf:RDF>
```

Context-aware adaptation service tier:

This tier is composed of at least one CAAS servers that can support migration of agents, execution of back-end modules, and adaptation of applications. Any two CAAS servers connect with each other via Remote Method Invocation (Java RMI) and IP multicasting. A RMI connection built to serialize objects and to transmit the serialized objects over networks, can provide agent migration and remote procedure call. Through the connection, personal agents can carry their owners' back-end modules and migrate from one CAAS server to another. Depending on different considerations of applications, there are two modes (synchronous/asynchronous) designed to control the conjunction between a user and his personal agent. Setting synchronous mode leads the personal agent to migrate with its owner. However, the agent anchors at its resident server when asynchronous mode is set; the personal agent will not migrate wherever its owner arrives.

Nevertheless, how to construct the connection between the servers is a problem. In our system, we exploit IP multicast among servers to make each server listen to the same IP multicast address. Accordingly, a common communication channel is formed for transmitting multicast messages between the servers. On the channel, there are two types of messages transmitted in this system. One of them is the message, which is sent for constructing RMI links between any two servers; the other is the one that is received by all servers for broadcasting notifications of deployment of applications. In implementation, the multicast connection is constructed at the initiation of this system; namely, all servers have the connection before constructing RMI connections. Thus, a server can build a RMI channel to connect with another by broadcasting a joining message to other servers through IP multicast. Servers receiving this message will reply with its IP address/port and server information. Afterwards, upon the server

receives a reply, it will construct a connection to the servers that replied messages.

Furthermore, CAAS servers are capable of performing application adaptation– a process to decide proper components for the carried application– on the applications carried by the agents migrated from other servers. An image transformation function, for instance, can be implemented by two candidate components: BMP-to-PNG and BMP-to-JPEG components. The former will be applied when the requesting client device is a J2ME-capable phone. While the user uses a desktop computer instead, the latter component can be chosen to implement this function.

A CAAS server principally includes the four constituents: *client interaction agent*, *context manager*, *agent manager*, and *component manager*. *Client interaction agent* serves as an interactor, which communicates with CC/PP negotiators on user devices, can transmit messages of the CC/PP negotiation protocol and those of remote dynamic invocation.

For communicating with user agents on client devices and adapting application based on contextual information, *context manager* plays the role to negotiate initial registration with the CC/PP protocol and to parse the embedded CC/PP profiles further. To handle these profiles, we exploit DELI service [33] and Jena API [34] to implement two sub-components of a context manager. One is *request solver*, which is capable of unpacking HTTP1.1 request messages to retrieve the CC/PP profiles; the other is *profile processor*, which can parse the CC/PP profile. In addition, all of the parsed profiles will be collected into *profile cache*; thus, devices can transmit the changed part to the service instead.

Agent manager, in a server, is capable of constructing personal agents and maintaining these agents. Additionally, it can control an agent's lifecycle, and invoke the corresponding method related to state change of the lifecycle. In order to save the

cost of constructing agents, and to immediately respond to user devices when reconnecting soon, *agent pool* is built to store agents constructed beforehand, and recycle the appointed ones back, respectively.

Component manager supplies the agent manager with the classes needing to be constructed in application adaptation. The reasons for constructing classes and the procedure will be explained in Section 4.4. Figure 23 presents the internal classes for creating the object instances to construct agents, applications, and components by `createAgent()`, `createApplication()`, and `ceateComponent()` respectively (explained in Section xxx). Also, an `ApplicationContext` class, which contains *application structure table*, *component decision tree*, and *changed component table*, etc, can support application adaptation and component construction.

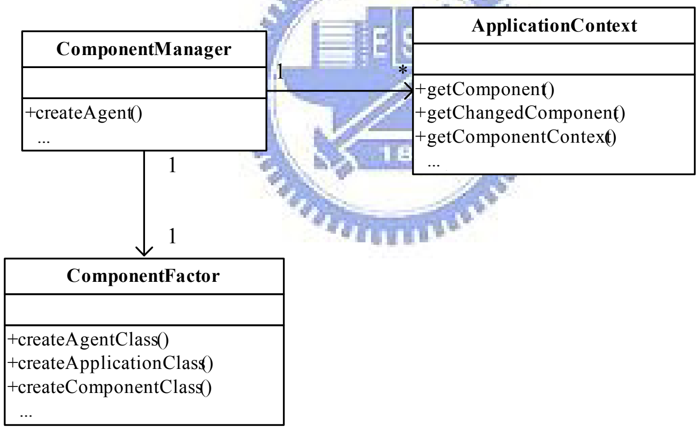


Fig. 23 Classes of component manager, component factory, and application context

Between any two CAAS servers, the connections constructed by *server communication agents* include an IP multicast channel and an RMI connection. Two CAAS servers can use the IP multicast channel to build a RMI connection for offering agent migration and remote method invocation.

Data access agent has the function of requesting and receiving data from the repository service. Between a CAAS server and the repository service, except for the

IP multicast, there is a connection built by using `URLClassLoader`. `URLClassLoader` is a Java class capable of loading classes from remote computers; it will be used if the required classes do not exist in the creation of the user-defined classes.

Repository service tier:

This tier plays the database and directory service role in our system. Three categories of information are stored. They include application information, personal information and context profiles, shown in Figure 22. Additionally, it stores the classes and ASCC files of the applications deployed. Note that an *Application Structure & Component Constraints* (ASCC) profile, designed in this framework, is an XML-based profile to describe application structures and component information. When a programmer has finished developing an application, he can pack the code of the application into a Java Jar file (a compressed file containing the class files), and stores this packed file and the ASCC description of the application into the repository service.

In this system, in order to deliver the information concerning a deployed application to CAAS servers, two mechanisms are designed. One is *application preloading*, in which the repository service notifies the CAAS servers once an application is deployed into the repository; the other is *application remedy*, that can be applied when a CAAS server accepts a migrated agent, but the required classes of the applications carried by the agent are not found. It will be detailed in Section 4.4.

4.1.4 Agent migration

The state transfer of the agent

A personal agent, which is an active object with a state, is assigned to serve a user. The term "active" means that the agent has a thread to perform a certain method invocation requested by the front-end module. An agent will invoke the requested method of the back-end side when it receives a request. Consequently, the result is sent back to the front-end modules. The state transition of an agent is presented in Figure 24.

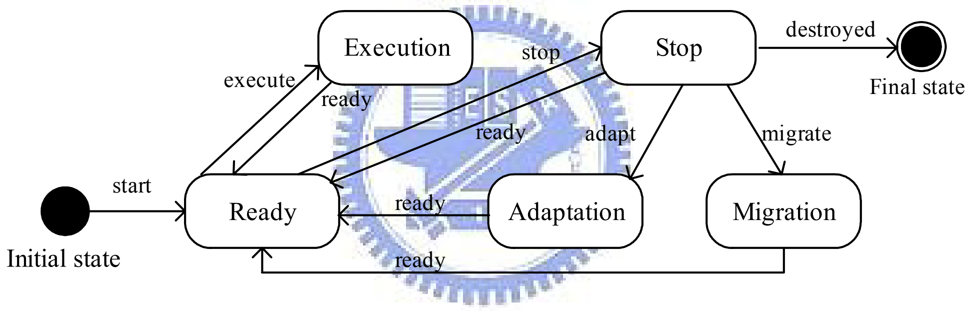


Fig. 24. State transition diagram of the personal agent

In the `Ready` state of an agent, the agent is activated to be ready for receiving invocation request from the front-end module. When receiving a request, it invokes the corresponding method of the appointed application. Then, the state will transit to the `Execution` state. When invoking is completed, the agent will send the result of the execution to the front-end module, and its state will change back to the `Ready` state. Moreover, the carried applications can be adapted by the agent manager only in this state. If this occurs, it will change to the `Stop` state and to the `Adaptation` state soon afterwards. In the `Stop` state, the agent is deactivated and does not receive any requesting invocation. Thereafter, components of each function of the application can be switched appropriately. The situations that cause the state to transit to the `Stop` state are: (i) a logout message received from the user agent; (ii) no messages received from the user agent for a period of time; (iii) the agent is instructed to migrate to another server. Moreover, conditions that make the state transit back to the `Ready` state are: (i) an agent manager has got the agent from the agent pool and then assigned it to its user for recycling; (ii) application adaptation has been performed; (iii) agent migration has been completed.

The structure of an agent

In this section, we discuss the inner structure of an agent. Figure 25 indicates an agent, which is composed of a state and a body. The state records the information related to the agent's user and the applications carried by the agent.

Agent State

An agent state (Figure 25) is composed of *agent ID* (the identifications of the agent), *User ID* (its owner), *Device ID* (the owner's device), and *Application IDs* (the applications carried). In addition, the agent state records the states of applications. Each of the states includes *absent component IDs* and *an event queue*. Absent component IDs are the identifications of the component objects withdrawn from the agent body. An event queue is responsible for queuing the requested events in the execution state. The queue is used to keep events, so it stops the execution of processes from being interrupted by incoming events. Specifically, an event queue retains the events of the state transition and notifications of the invocations from the front-end module. State transition and invocations will be scheduled in the FIFO order,

so if a new event arrives, it will be put at the rear of the queue. Then, to process these events, the main thread of this agent obtains an event from the front of this queue. Taking the scenario in Section 1.1 for example, the user agent, in the front-end module of a user device, requests the user's personal agent for a picture. When the personal agent receives this invocation, events concerning the invocation will be generated and put in the queue. In this example, the events corresponding to 2, 3, 4, 5, 6, 12 are put into the event queue. Next, if any request arrives or the state transition is triggered, the notification relevant to these events will follow the previous requested notifications.

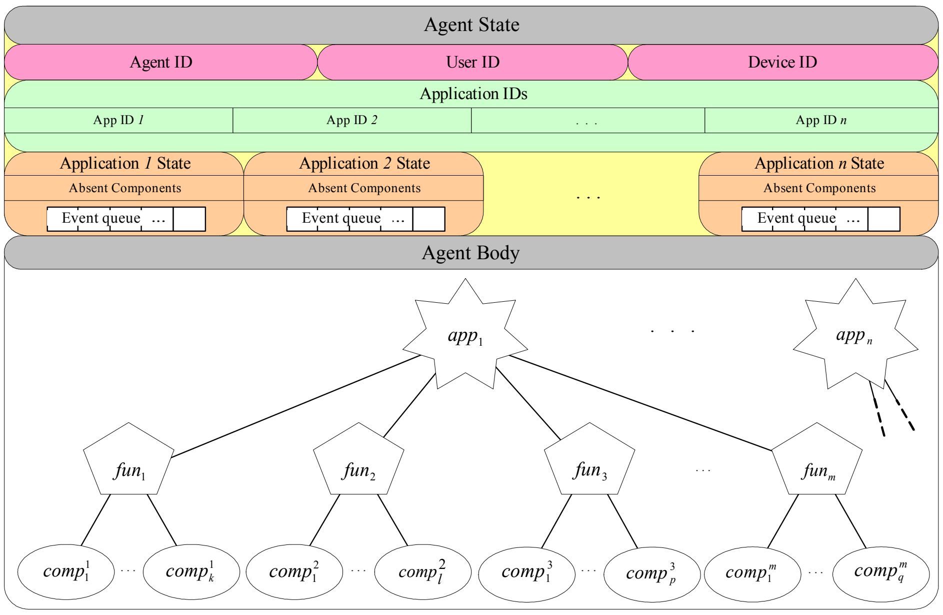


Fig. 25. An internal view of a personal agent and an application structure

Agent migration

In the application ImageGathering, even when moving from room to room, users can continue collecting the information. In order to complete this, we need to overcome the following problems: "How does the system perceive the situations of users' movement?" and "How does the system instruct an agent that serves the user to

migrate with the user under perceiving the situations of users' movements?".

(1) Types of mobility

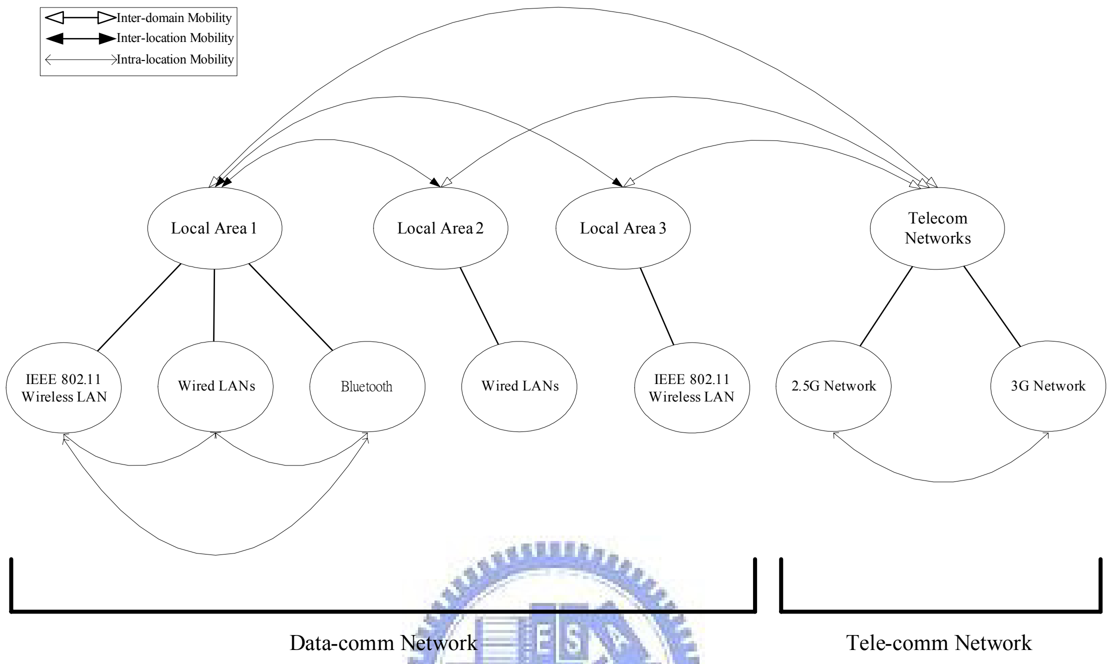


Fig. 26. Cases of users' movements

We can roughly partition off networks into data-com network and tele-com network. Data network includes IEEE 802.11x [35, 36], wired LANs, and bluetooth networks [36]. Tele-com network consists of 2.5/3G [37] networks (Figure 26). According to characteristics of these networks, we define personal mobility and terminal mobility. Personal mobility means that by using any nearby computing equipment, a user does not need to carry his device wherever he moves. In other words, a user can use a device to perform his work, and also continue working via another instead. Terminal mobility indicates that a user can perform his work via his carried device.

As shown in Figure 26, the data-com network contains a great many local area networks (LANs), and the three kinds of networks may be in the same region, as Local Area 1. Furthermore, there is one possible type of network in a LAN, such as Local Area 2. In a tele-com network, numerous wireless tele-com network areas, formed by the radio coverage of base stations, are regarded as the same network in our system.

The cases of users' movements from one region to another can be grouped into three categories: *Intra-location mobility*, *Inter-location mobility*, and *Inter-domain mobility*. *Intra-location mobility* means that the coverage of a user's movement does not exceed the range of a LAN. For instance, when a student collects information through his personal computer in his lab. *Inter-location mobility* indicates that a user's movement crosses two LANs. A case of this movement might be that a user uses a certain device in Local Area 1, and then uses another after moving to Local Area 2. *Inter-domain mobility* refers to the fact that a user's movement crosses data-com networks and tele-com networks. A student, for example, collects images by using his personal computer in his lab. Next, in place of the personal computer he uses a Java Phone when moving from his lab to a meeting room.

(2) Agent registration

The system provides two mechanisms to perceive users' movements. We call the first mechanism *Passive-Client and Active-Server* (*PCAS*). In the mechanism, the user agent of a user's device will be notified to initiate a registration procedure when its user moves to the server's covered region. We call the second mechanism *Active-Client and Passive-Server* (*ACPS*). By using this mechanism, user agents of a user's device will actively inform the repository service if they need to connect to some CAAS server.

In PCAS, a server located in a region can detect movements of user devices entering into this region. When a user uses his device and enters this region, the server notifies the user agent on his device. Thereupon, the user agent will register with this server. Intrinsically, notification messages are the advertisements broadcasted

periodically on the wireless IEEE 802.11 network by a CAAS server. User agents of client devices continue listening to this kind of messages. Provided that there is a user entering a new wireless LAN, then the user agent will send a requesting service message to the server sending the notification without registering to any server. Figure 27 illustrates the sequence diagram of this procedure. To inform the server of client information, we embed the CC/PP profile in the request service message. While a server receives the request message from the `ClientInterActionAgent` object, it will forward the messages to the `RequestResolver` object to resolve the CC/PP profile. The `RequestResolver` object is capable of retrieving the profile from the message and passing it to the `ProfileProcessor` object to resolve the profile. Then, the result will be passed to the `CAAS` service object, `c2`. When receiving the message, `c2` requests CAAS service `c1` for the user's personal agent. The user's personal agent, therefore, can be instructed to migrate to the server `c2` close to the user. In this mechanism, user agents on users' devices can automatically register to CAAS servers when their owners move among LANs.

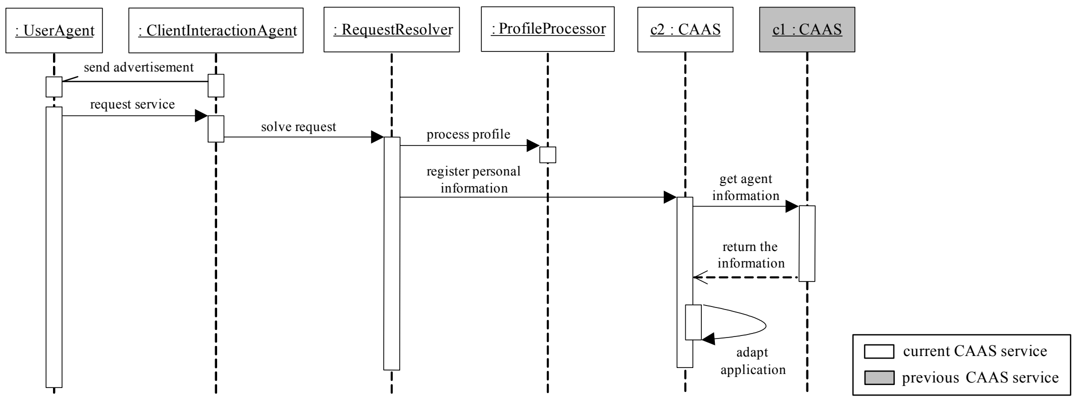


Fig. 27. The sequence diagram of ACPS

The main difference between ACPS and PCAS is that in ACPS user agents on devices actively register to the repository service. Thus, ACPS can be applied to solve the condition where user agents on user devices have not connected to any server. A user agent on a user's device, for example, sends a request message to the repository service. Upon receiving a request message, the repository service redirects the connection to the nearby CAAS server closest to him. This procedure is decomposed into steps shown in Figure 28.

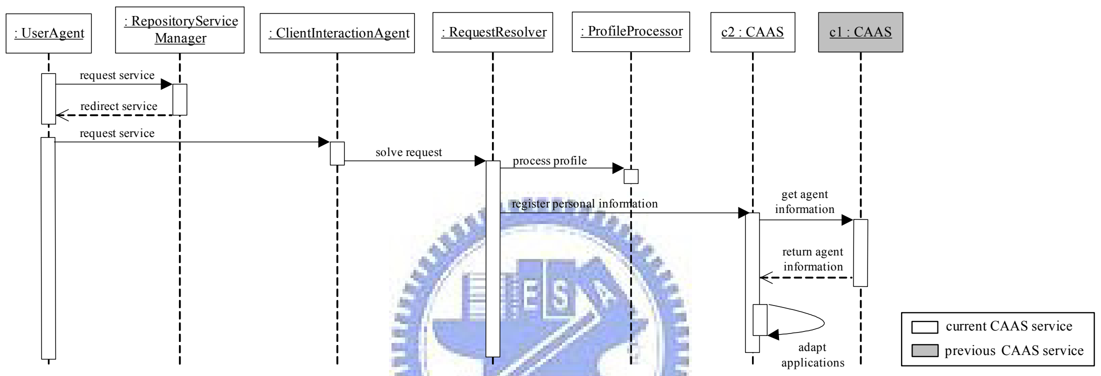


Fig. 28 The sequence diagram of ACPS with agent migration

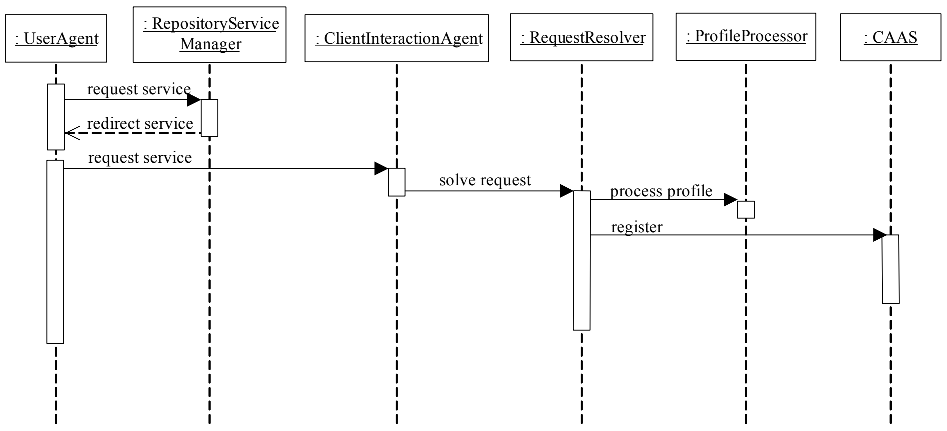


Fig. 29 The sequence diagram of ACPS without agent migration

It will be possible that agent migration is not needed if the covered regions of user movements are identical. Perhaps one of the conditions is that the device briefly disconnects, and then reconnects to the server. In this condition, the user's personal agent still resides in this server. Thus, when the user agent inquiries the repository service about a CAAS server, the repository service will inform the user device of the original nearby CAAS server, and redirect the connection to that CAAS server. Though the user device reconnects to the CAAS server, the personal agent will not be instructed to migrate.

We explain below how the system perceives users' movements and when a CAAS server instructs a user's personal agent to migrate from another server. Further, the cases of users' movements can be considered altogether with PCAS and ACPS, arranged below:

- In Inter-domain mobility, two situations are classified into this category. One condition is that the underlying network accessed is data-com network first and tele-com network subsequently. In the condition, user agents on user devices can register automatically by using ACPS. The other is the opposite of the first condition. Here user agents on user devices can be notified to register through PCAS.
- In the cases of Inter-location mobility where a user crosses two different kinds of local area networks, user agents on user devices can be notified to register through PCAS.
- Under the conditions of Intra-location mobility, it is unnecessary to move users' personal agents, because in this case users use their devices on the same network.

(3) Agent migration strategy

In agent migration, we consider the three strategies: Heavyweight Agent Migration

(HAM), Flyweight Agent Migration (FAM), and Lightweight Agent Migration (LAM).

Heavyweight Agent Migration (HAM):

In the Heavyweight Agent Migration (HAM) strategy, an agent will carry the components belonging to Type 2 and Type 3 while migrating, except for those specified "`carried='No'`"[1] in the ASCC profile. (1) of Figure 30 illustrates an example. Agent migration is a procedure that serializes object instances comprising a whole agent into a byte array, and then sends the serialized binary data to the target server. Upon receiving it, the receiving server reconstructs the agent from the byte array.

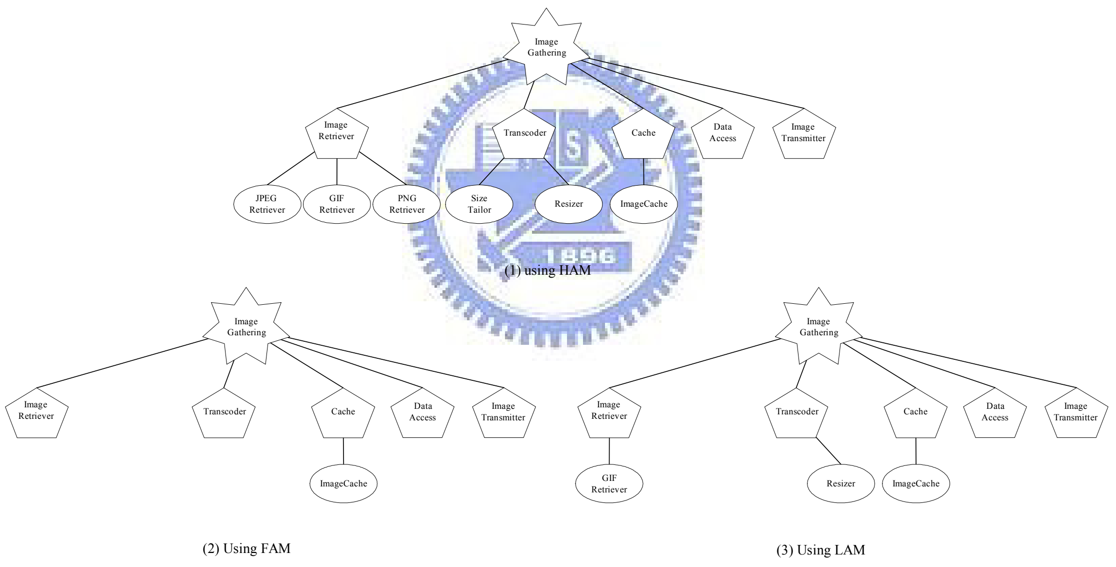


Fig. 30. The structure of the application ImageGathering by using three strategies.

Suppose that an application is carried with two functions: *fun 1* and *fun 2*. *fun 1* can be implemented by components *comp 1* and *comp 2*, and *fun 2* can be realized by *comp 3* and *comp 4*. Figure 31 exhibits the results of the HAM strategy applied to agent migration. The components *comp1-4* are carried with agent migration. When the agent reaches CAAS server 2, the server appropriately adapts the applications carried

[1] The term "not carried" means nullifying the object references in the implementation code.

by the agent. For function *fun 1*, the component *comp 1* is suitable for the context of the user device used previously. However, it is not suitable for that of the other used subsequently. As a result, the component *comp 2* is chosen to substitute *comp 1*.

Listing 11 shows the HAM algorithm. At the transmitter side, the immoveable components are detached and then their IDs are recorded into an absent component array in an agent state *S*. While accepting the agent on the receiver side, the receiving server will retrieve the application IDs from the agent state and recorded them into array *A* (Lines 1-2). If, for each function, a certain component implementing this function is unsuitable, another proper component will be chosen to substitute for that component by means of *DECIDE-PROPER-COMPONENT(A[i], F[j], T, Q, D)*. Eventually, Line 16 switches each unsuitable attached component to a more appropriate one for each function *F[j]* of an application *A[i]* in the agent *G*.

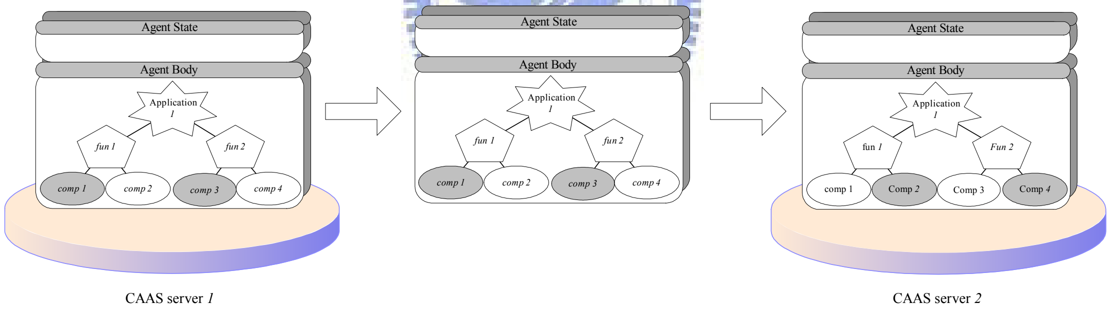


Fig. 31. Agent migration using HAM

Listing 11. The algorithm of HAM

G: an agent; *S*: the agent state of the agent *G*; *A*: an array recording IDs of the applications carried by the agent *G*; *F*: functions comprise an application *A[i]*; *T*: a previous context profile; *Q*: a current context profile; *C*: an array recording IDs of un-carried components of an application *A*; *D*: a decided components for functions *F* of an application *A[i]*

```
HAM-TRANSMITTER($G$)
1 GET-AGENT-STATE($G,S$)
2 GET-APPLICATION-IDs($S, A$)
3 for $i \leftarrow 0$ to $length[A]$
4 do GET-FUNCTION-IDs($S, A[i], F$)
5     for $j \leftarrow 0$ to $length[F]$
6     do DETACH-IMMOVEABLE-COMPONENT($G, A[i], F[j], C$)
8         for $k \leftarrow 0$ to $length[C]$
8         do ADD-TO-ABSENCE-COMPONENT($G, S, A[i], F[j], C[k]$)
9 return $G$


HAM-RECEIVER($G, T, Q$)
10 GET-AGENT-STATE($G,S$)
11 GET-APPLICATION-IDs($S, A$)
12 for $i \leftarrow 0$ to $length[A]$
13 do GET-CHANGED-OR-ABSENCE-FUNCTION-IDs($S, A[i], F$)
14     DECIDE-PROPER-COMPONENT($A[i], F, T, Q, D$)
15     for $j \leftarrow 0$ to $length[D]$
16     do SWITCH-COMPONENT-TO($G, A[i], F[j], D[j]$)
17 return $G$
```

Flyweight Agent Migration (FAM):

The principle of this strategy is to minimize the data size needed to transfer an agent between servers. In other words, the components, except for those classified to Type 2 and Type 3 and indicated as "`carried='Yes'`", are not carried with agent migration. Figure 30 (2) illustrates this example since, except for the component Image Cache, no component is carried with agent migration. This is because Image Cache is Type 2 and declared "`carried='Yes'`", but the other components are the cases in either Type 1 or Type 2/3 components declared "`carried='No'`" (see Listing 12). Figure 32 illustrates that the components *comp 1-4* are not carried since

these components are classified to Type 2/3 but not specified "carried='Yes". Then, in CAAS server 2, for each function proper components are decided. Additionally, their object instances are reconstructed if necessary.

The algorithm of FAM is shown in Listing 12, where components, except for the Type 2/3 components specified "carried ='Yes'", are not carried in agent migration. At the acceptance of the agent, the server can examine the missing components (retrieved into array *F*). After deciding proper components for each function in Line 14, the receiver creates object instances for those in *F*, and plugs the suitable ones into their corresponding functions (in Lines 15-16). However, there is likely to be a problem: the required classes (the user-defined subclasses of `Component`) are not found. If this problem occurs after an agent migrates, the needed classes can be loaded through the component manager. Details for this will be specified in Section 4.4.

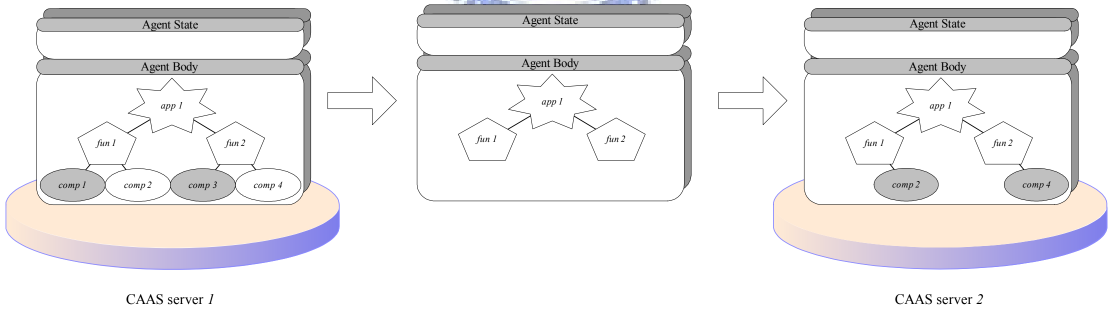


Fig. 32 Agent migration using FAM

Listing 12. The algorithm of FAM

```
FAM-TRANSMITTER($G$)
1 GET-AGENT-STATE($G,S$)
2 GET-APPLICATION-IDs($S, A$)
3 for $i \leftarrow 0$ to $length[A]$
4  do GET-FUNCTION-IDs($S, A[i], F$)
5      for $j \leftarrow 0$ to $length[F]$
6       do DETACH-ALL-EXCEPT-CARRIED-COMPONENT($G, A[i], F[j], C$)
7           for $k \leftarrow 0$ to $length[C]$
8            do ADD-TO-ABSENCE-COMPONENT($G, S, A[i], F[j], C[k]$)
9 return $G$


FAM-RECEIVER($G, T, Q$)
10 GET-AGENT-STATE($G,S$)
11 GET-APPLICATION-IDs($S, A$)
12 for $i \leftarrow 0$ to $length[A]$
13  do GET-CHANGED-OR-ABSENCE-FUNCTION-IDs($S, A[i], F$)
14      DECIDE-PROPER-COMPONENT($A[i], F, T, Q, D$)
15      for $j \leftarrow 0$ to $length[D]$
16       do ATTACH-COMPONENT-TO($G, A[i], F[j], D[j]$)
17 return $G$
```

Lightweight Agent Migration (LAM):

The substance of this strategy is that one component is carried for each function in an application, except for Type 1 components, when agents migrate. A possible method we propose is to carry the only components which implement its corresponding functions in agent migration. (3) of Figure 30 illustrates this strategy, where only the components implementing their corresponding functions are carried. Except for Type1 components (Data Access and Image Transmitter), the components of the functions Image Retriever, Transcoder, and Cache are carried.

Likewise, in Figure 33, before the agent migrates, the components *comp 2* and *comp 4* are detached from the functions *fun 1* and *fun 2* respectively. While the agent arrives in CAAS server 2, *comp 2* and *comp 4* are chosen as the proper components for *fun 1* and *fun 2*, individually. Therefore, instances of the two components will be

reconstructed, and then attached into their corresponding functions.

Listing 13 presents the algorithm. At the transmitter side, Line 6 detaches all components, except for the Type 2 and Type 3 components implementing their functions. Their IDs are recorded to an agent state S (Line 8). When accepting the agent, the proper components will be determined to substitute for the components that are absent or unsuitable, as shown in Lines 13-16.

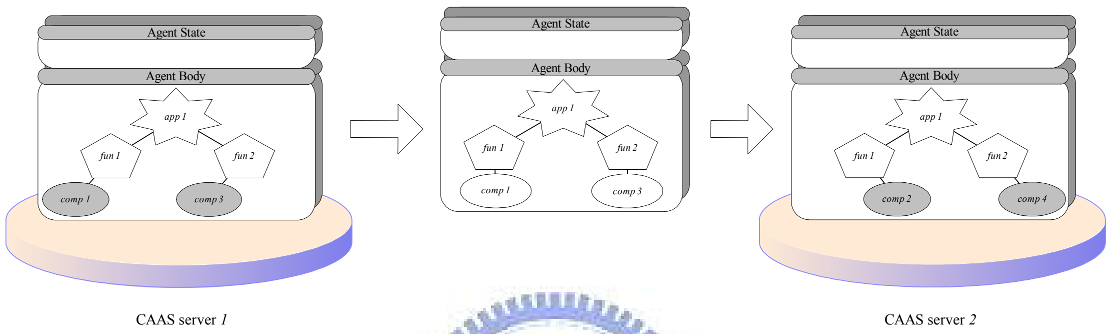


Fig. 33 Agent migration using LAM

Listing 13. The algorithm of LAM

```
LAM-TRANSMITTER($G$)
1 GET-AGENT-STATE($G,S$)
2 GET-APPLICATION-IDs($S, A$)
3 for $i \leftarrow 0$ to $length[A]$
4   do GET-FUNCTION-IDs($S, A[i], F$)
5      for $j \leftarrow 0$ to $length[F]$
6        do DETACH-ALL-EXCEPT-IMPLEMENTING-COMPONENT($G, A[i], F[j], C$)
7           for $k \leftarrow 0$ to $length[C]$
8             do ADD-TO-ABSENCE-COMPONENT($G, S, A[i], F[j], C[k]$)
9 return $G$

LAM-RECEIVER($G, T, Q$)
10 GET-AGENT-STATE($G,S$)
11 GET-APPLICATION-IDs($S, A$)
12 for $i \leftarrow 0$ to $length[A]$
13   do GET-CHANGED-OR-ABSENCE-FUNCTION-IDs($S, A[i], F$)
14      DECIDE-PROPER-COMPONENT($A[i], F, T, Q, D$)
15      for $j \leftarrow 0$ to $length[D]$
16        do SWITCH-COMPONENT-TO($G, A[i], F[j], D[j]$)
17 return $G$
```

Comparison

In this section, we tested these three strategies to see how the size and number of components affect time cost (msec) of agent migration and application adaptation. In the experiments, we consider HAM, FAM and LAM under the worst case. In addition, in LAM we measure cases of LAM under the best case. The best case means that all components carried by an agent do not need to be replaced. The worst case indicates that all components carried by an agent need to be switched to the proper ones. Figure 34 shows the experimental setting. We measure the round trip time during which CAAS server 1 informs CAAS server 2 to instruct an agent to migrate successfully. To analyze the results accurately, we measure each case for 10,000 times to compute the average of the results.

We experiment on the strategies through two measurements. First, we let an agent carry an application containing one function, which is implemented by two components. We consider the cases of HAM, FAM, LAM-B, and LAM-W by gradually increasing the size of the two components from 512 to128k Bytes. Table 7 and the left-hand side of Figure 35 demonstrate the time cost (msec) of the cases. As we can see, FAM performs worse than the other three; on the whole the cases of LAM-B and LAM-W cost less than the others, and those of LAM-B win. Second, we let an agent carry an application composed of one function, which can be implemented by 50, 75, 100, …, 250 components separately. In Table 8 and the right side of Figure 35, the results indicate that HAM and FAM perform worse than LAM-B; while LAM performs better than others. In the situation, the time needed increases with an increasing number of components. This is because each of the algorithms performs a certain operation one by one for each component attached. For

example, the HAM algorithm detaches all of the immoveable components.

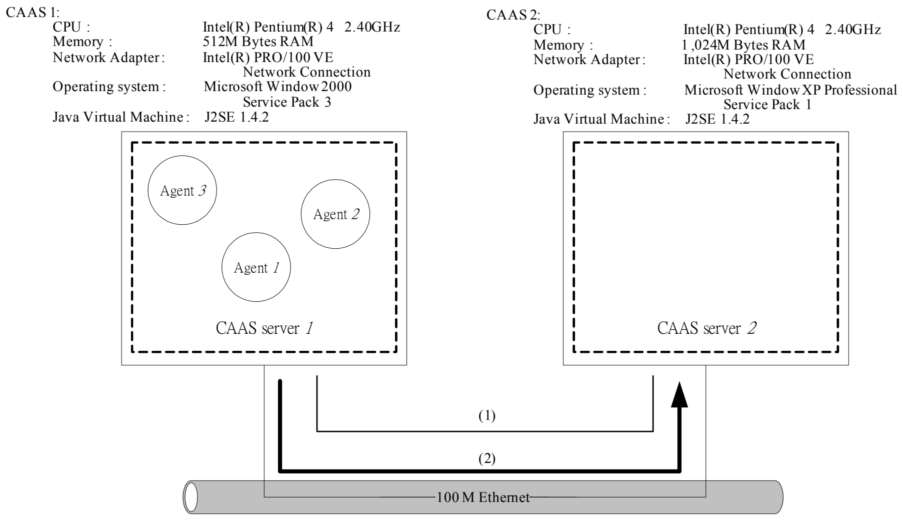

CAAS 1:
CPU : Intel(R) Pentium(R) 4 2.40GHz
Memory : 512M Bytes RAM
Network Adapter: Intel(R) PRO/100 VE Network Connection
Operating system : Microsoft Window 2000 Service Pack 3
Java Virtual Machine : J2SE 1.4.2
CAAS 2:
CPU : Intel(R) Pentium(R) 4 2.40GHz
Memory : 1,024M Bytes RAM
Network Adapter: Intel(R) PRO/100 VE Network Connection
Operating system : Microsoft Window XP Professional Service Pack 1
Java Virtual Machine : J2SE 1.4.2
Agent 3
Agent 2
Agent 1
CAAS server 1
CAAS server 2
(1)
(2)
100 M Ethernet

Fig. 34 The experimental setting

Tab. 7. The results of the first measurement

(one application, one function, two components)

Component Size (Byte)	**Heavyweight Agent Migration (HAM)**	**Flyweight Agent Migration (FAM)**	**Lightweight Agent Migration-B (LAM-B)**	**Lightweight Agent Migration-W (LAM-W)**
512	3.134	3.195	3.027	3.125
1024	3.345	3.409	3.249	3.253
2048	3.911	4.136	3.633	3.890
4096	4.250	4.333	4.192	4.284
8192	6.514	6.663	6.431	6.483
16384	8.472	8.627	8.494	8.556
32768	15.153	14.463	14.158	14.380
65536	28.467	28.242	27.942	28.063
131072	124.181	122.134	121.713	122.29

Tab. 8. The results of the second measurement

(one application, one function, one component)

Component Number	**Heavyweight Agent Migration (HAM)**	**Flyweight Agent Migration (FAM)**	**Lightweight Agent Migration-B (LAM-B)**	**Lightweight Agent Migration-W (LAM-W)**
50	7.016	7.021	6.816	6.800
75	9.895	10.050	9.793	9.825
100	13.866	14.656	12.473	12.029
125	14.160	14.340	14.018	14.045
150	19.667	20.514	19.430	19.444
175	20.482	20.994	20.712	20.732
200	21.844	22.141	21.931	21.841
225	24.719	24.323	24.378	24.380
250	28.770	29.221	28.443	28.422

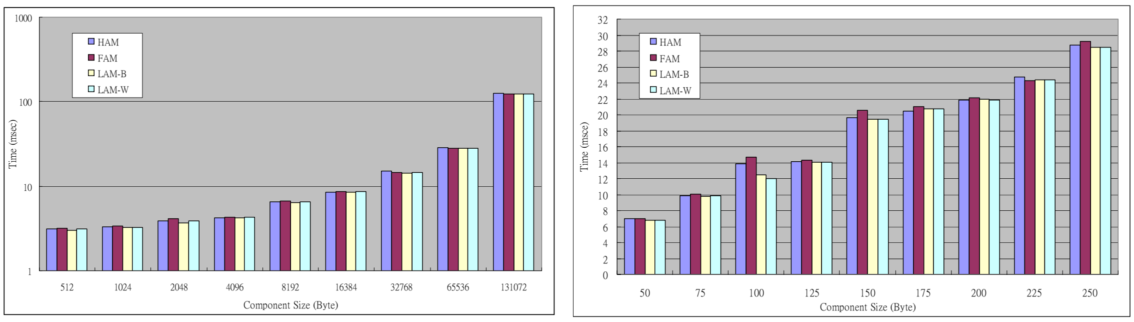


Fig. 35 The experimental results on HAM, FAM, LAM-B, and LAM-W

4.2 Ubi-Adapting

Figure 36 shows the implementation of our system. The framework contains two main parts: one is the server part and the other is client part. In the server side, we design Context Profile and Context Awareness Module used to be aware of the context of users. Application Profiles can describe the application structure. In other words, the module stores ASCC profiles of applications. There are personal agents which can migrate from one computer to the other. Besides, it can store its owner's information. In order to realize the function we explain in Chap 2 and Chap 3, we design the Application Adaptation Service and Representation Transformation Service to approach adaptation and transformation mechanisms. Figure 37 demonstrates the results we test.

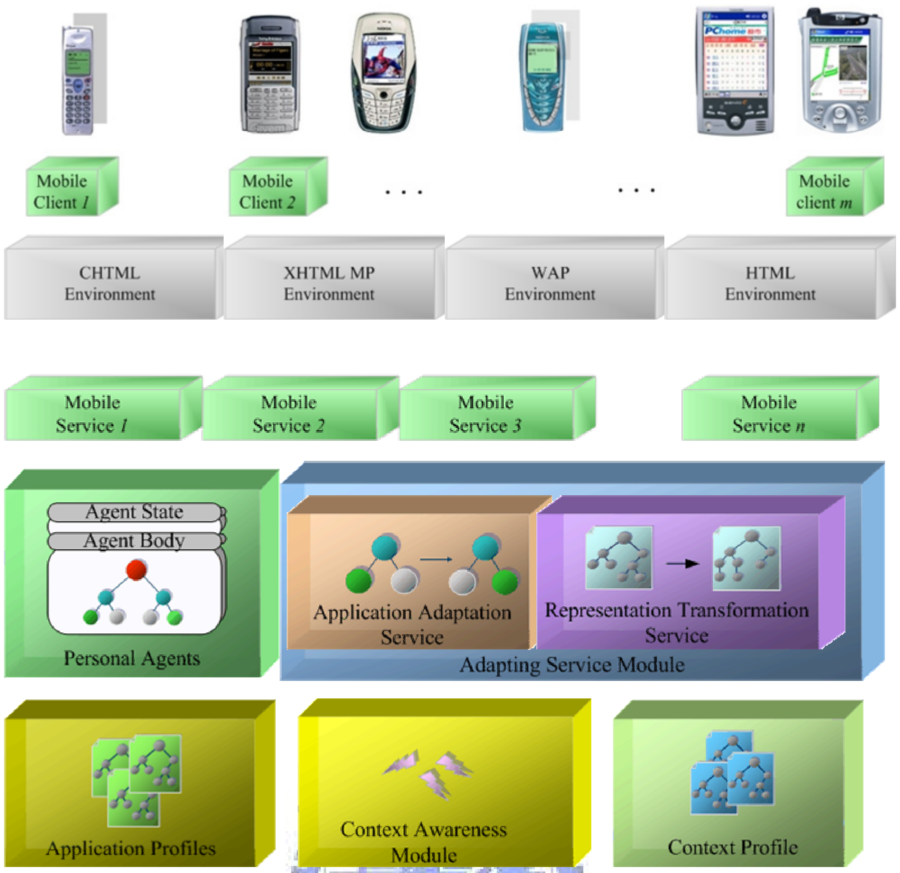


Fig. 36 The architecture of Ubi-Adapting

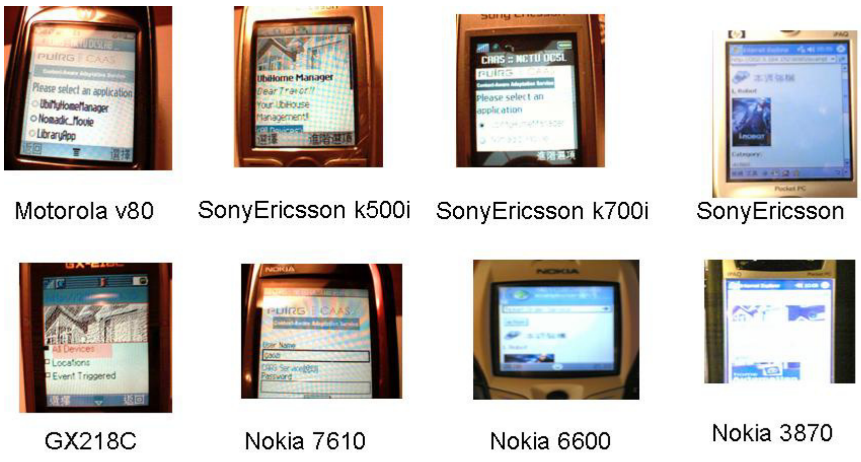


Fig. 37 the tested devices

4.3 Gateway of Gateway (G^2)

G^2 means ‘gateway of gateway.’ That is, there are several gateways in a home environment. These gateways include Home Gateway, TV Box, etc. Connecting these gateways through G^2, we can use various kinds of mobile devices. The components architecture of G^2 is similar to that of Ubi-Adapting. In addition, we design G^2 can detect coming bluetooth devices to identify the users around.

Figure 38, for example, demonstrates users can use the mobile devices and the computers to access the back-end services and the services implemented on the G^2. In this figure, we design a home control application. There are two functions we implemented. (1) Users can use different devices to control the home facilities. (2) Devices can be trigged due to the cause of coming users carrying bluetooth devices.

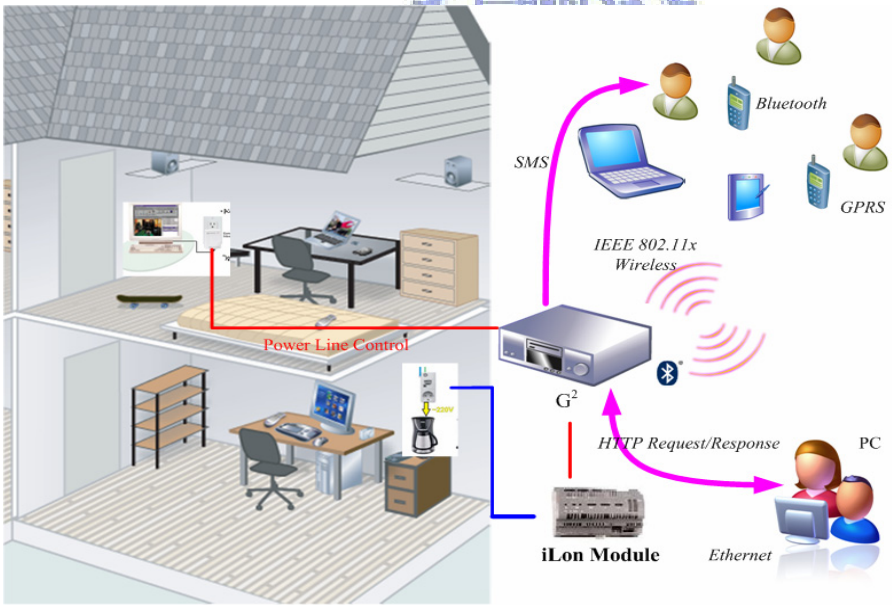


Fig. 38 the overview of the G^2

Tab 9 shows the revolution of the implemented systems. The differences between the two systems include the sensors we use or not, and which type of machine we used. In Ubi-Adapting, we don't use sensor, but use Bluetooth to capture the user's location. Besides, we use embedded box for the implementation of G^2.

Tab. 9 the revolution

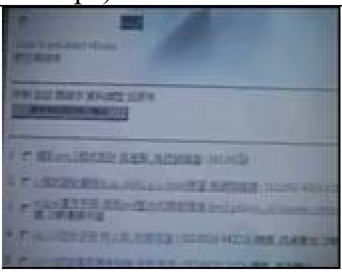
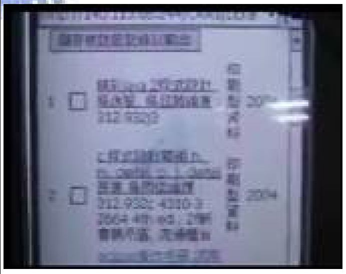
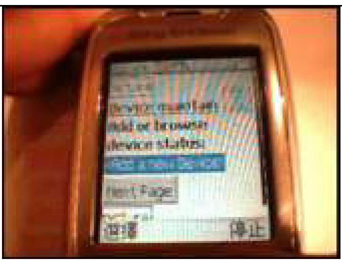
Name	**Ubi-Adapting**	**G^2**
Profile	CC/PP WAP UAProf	CC/PP WAP UAProf
Sensor	N/A	Bluetooth
Performer	PUML transformer Component adaptor	PUML transformer Component adaptor
Application	Application Migration	Application Migration
Decision Engine	N/A	N/A
Embedded Box	No	Yes
Application Practice	Library, Movie, Home control	SMS, Scheduling Home Control

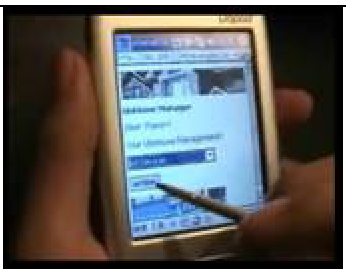
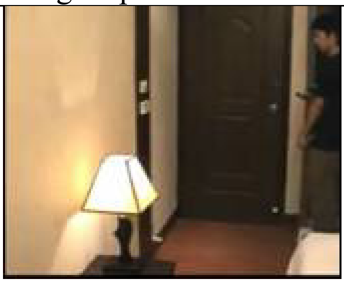
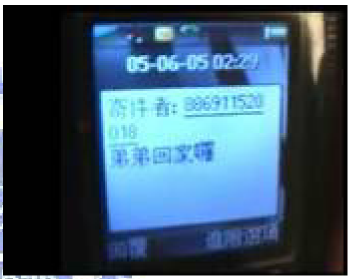
4.4 Applications

Table 10 we explain the application we implemented. A movie theater mobile web site is the first application. A User can use a WAP phone, and then change to use an XHTML MP phone (Nokia 6600 and Nokia 7610). For example, if a user uses a Nokia 7610, the downloading link with the high-quality movie clip will be shown in the page. If a user uses Nokia 6600, the downloading link shown is the low-quality movie clip. In the library application, users can use PC, Pocket PC 2002, and Nokia 7610. According to the context of the used device, the proper format of the content will be sent to the user's device.

Tab. 10 The application demos

Application	Description	Demo
A movie mobile web site	Users can use their mobile devices to order tickets, download clips, etc. Besides, the UI syntax (e.g. WML and CHTML, etc) and downloading clips are adapted to the contexts of devices at runtime.	using WAP phone to inquire hot movie using Nokia 7600 to download clip (MPEG4 , encoding rate: 64kbps)

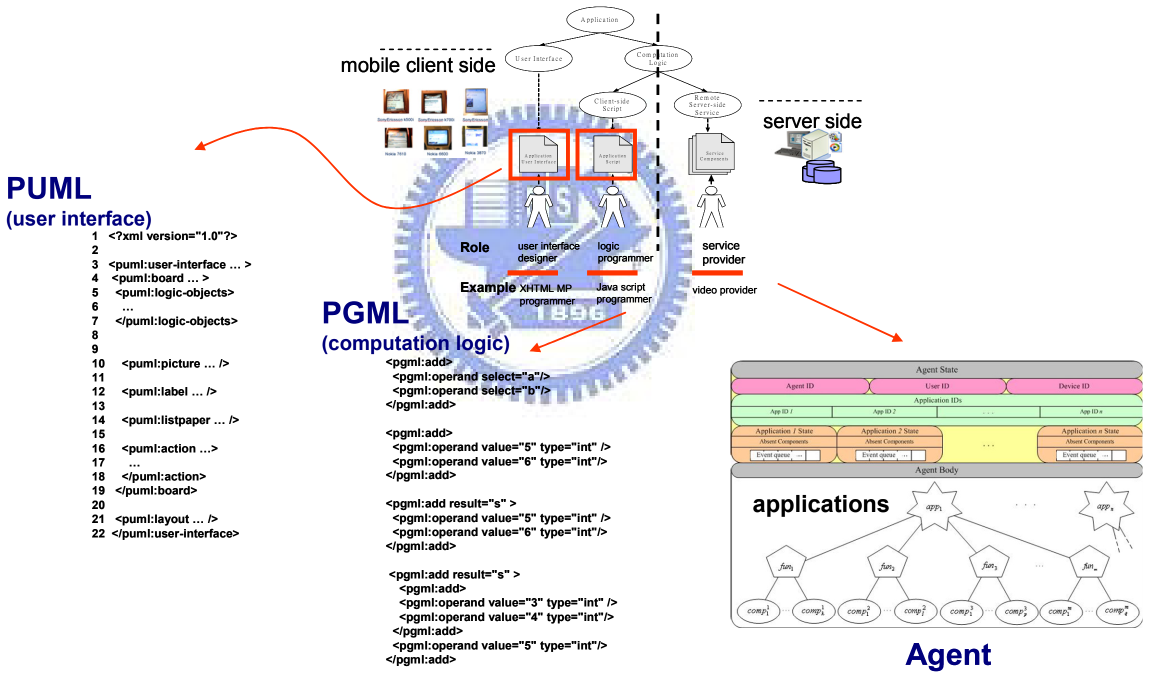
		using Nokia 6600 (3GPP, encoding rate: 32 kbps)
Unlimited Library	Users can use these mobile devices to inquire the books.	 using PC to browse the result using Nokia 6600  using Pocket PC2002
Mobile Home Service:	Several home networks can be exploited to control the home facilities. We use X10 plug-able modules in this application, and the mobile phones and PDAs to control the facilities.	 using SonyEricsson k700i to turn the power on

		 using Dopod 818
G^2 (Gateway of Gateway): an embedded home box	The G^2 has the following features: ● a CAAS server is customized to be deployed on a embedded box. (using EPIA x86 main board, shown below) ● Using Bluetooth to Identify people ● SMS message notification ● Multi-devices used to control home facilities	 Once senoring a coming family, the power is turned on automatically  Other family members receive SMS notification

Chapter 5

Application development

Figure 39 shows the programming model. There are three roles to?? who?? write the PUML/PGML files, components of applications and the agents used to carry the applications. The development flow is shown in Figure 40.



```
<?xml version="1.0"?>

<puml:user-interface ... >
 <puml:board ... >
  <puml:logic-objects>
    ...
  </puml:logic-objects>


   <puml:picture ... />

   <puml:label ... />

   <puml:listpaper ... />

   <puml:action ...>
     ...
   </puml:action>
  </puml:board>

  <puml:layout ... />
 </puml:user-interface>
```

```
<pgml:add>
  <pgml:operand select="a"/>
  <pgml:operand select="b"/>
</pgml:add>

<pgml:add>
  <pgml:operand value="5" type="int" />
  <pgml:operand value="6" type="int"/>
</pgml:add>

<pgml:add result="s" >
  <pgml:operand value="5" type="int" />
  <pgml:operand value="6" type="int"/>
</pgml:add>

 <pgml:add result="s" >
   <pgml:add>
   <pgml:operand value="3" type="int" />
   <pgml:operand value="4" type="int"/>
  </pgml:add>
  <pgml:operand value="5" type="int"/>
</pgml:add>
```

Fig. 39 PUML/PGML & agent development

Figure 40 contains the three steps: (1) PUML/PGML writing, (2) application component and agent development, (3) application deployment. As we can see in this figure, programmers can edit PUML and PGML files directly (e.g. Edge 1.1) or use the toolkit to generate the files (Edge 1.2).

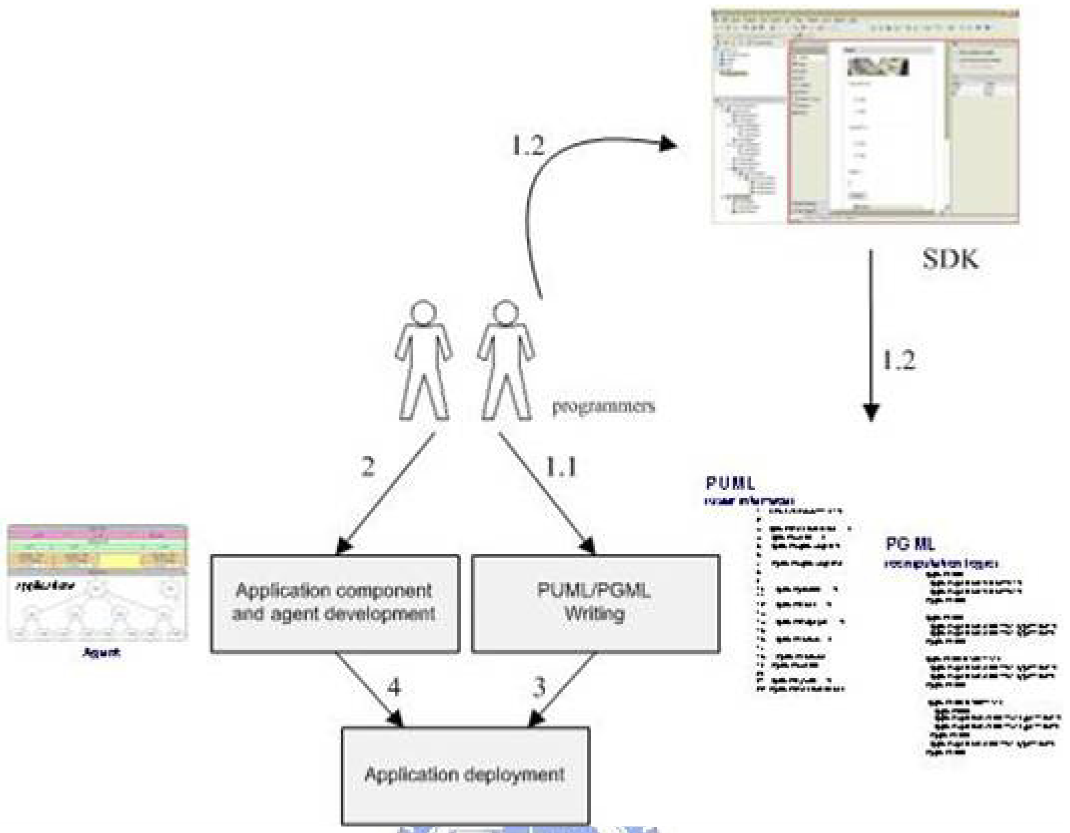


Fig. 40 the steps for development

The following sections explain the steps: (1) PUML/PGML writing in Sec 5.1, (2) application component and agent development in Sec 5.2, (3) application deployment. In PUML/PGML writing, programmers can construct PUML/PGML pages. Besides, programmers can use a SDK to generate the page in drag-and-drop manner. In the application, we can use the generated page. The PUML/PGML page can be transformed at runtime. When designing applications, programmers can write the code capable of transforming the PUML/PGML document into the target languages.

5.1 PUML/PGML writing

5.1.1 Hand coding

As we can see in Figure 40, there is a step to write PUML/PGML document. In the section, we will demonstrate an example written in PUML and PGML documents, and the simulating results of the generated J2ME and WML codes by applying the transformation mechanism mentioned above. Listing 14 is a PUML document (UIExample.puml). There are three `<puml:textnote>` elements in the listing. The first two can get two input numbers, and the last can display the result by summing the two numbers up. The `<puml:action>` … `<puml:action>` block describes that the two inputted values are passed into the `sum` method of `object1`, e.g. `addTwoNum.pgml` (declared in Line 12-14). Furthermore, the value attribute of the `<textnote>` widget, `sum` in `board2`, will be updated by the retuned value after the action is triggered. Line 28 describes the code to accomplish that. The section of `addTwoNum.pgml` can sum the two input number, shown in Listing 7 in Section 3.2.2.

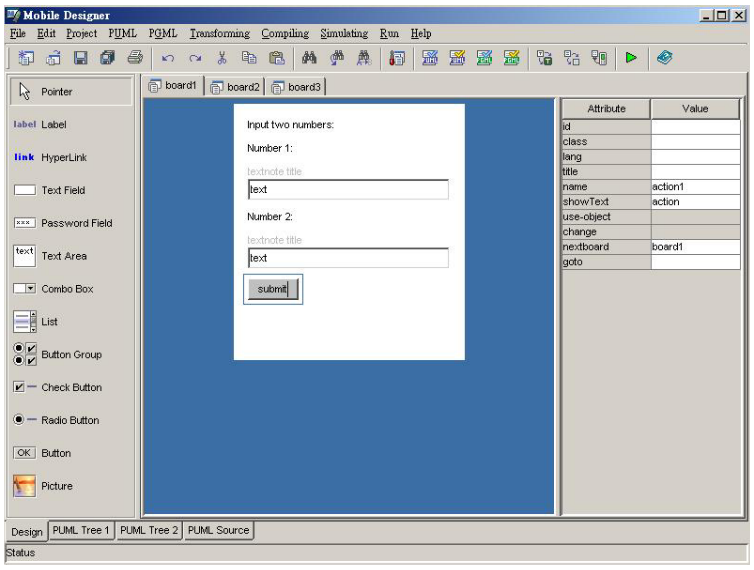
Listing 14. A user interface described in PUML

```
1  <?xml version="1.0"?>
2  <puml:user-interface name="UIExample" version="1.2"
3   xmlns:puml="http://dcsw3.cis.nctu.edu.tw/Project/
4   pervasive/PUML/">
5
6    <puml:board name="board1" title="FirstPage" >
7      …
8    </puml:board>
9
10   <puml:board name="board2" title="SecondPage">
11
12     <puml:logic-objects>
13       <puml:object name="object1"  source="addTwoNum.pgml" />
14     </puml:logic-objects>
15
```

```
    <puml:label name="mainTitle" showText="Input two numbers:" />

    <puml:label name="num1Title" showText="Number 1:" />
    <puml:textnote name="num1" value="0" />

    <puml:label name="num2Title" showText="Number 2:" />
    <puml:textnote name="num2" value="0" />

    <puml:label name="sumTitle" showText="Sum:" />
    <puml:textnote name="sum" value="0" />

    <puml:action name="action2" showText="action2">
      <puml:change container="board2" component="sum" update="value">
        <puml:use-object name="object1" method="sum">
          <puml:param select="num1" />
          <puml:param select="num2" />
        </puml:use-object>
     </puml:change>
    </puml:action>

  </puml:board>

</puml:user-interface>
```

5.1.2WYSIWYG

Besides hand coding, programmers can use an SDK to generate PUML and PGML documents to accelerate the development, shown as in Figure 42. Programmers can use the toolkits to develop web page in drag-and-drop manner.



Attribute	Value
id	
class	
lang	
title	
name	action1
showText	action
use-object	
change	
nextboard	board1
goto	

Fig. 41 using the toolkit for development

We display the design in Figure 41. It is the layout view for code in Listing 15. Besides, there are two versions of toolkits we design. One is the toolkit we embedded the drag and drop function into the some famous SDKs. In our design, we embedded it into Borland JBuilder[38] and Microsoft .NET studio[39]. The two toolkits are popular currently. We can leverage them to promote our framework. The other is the web-based toolkit. Programmers can design pages by using the user interface of browsers, such as IE, firefox, etc. In the figure 41, there JBuilder version can generate the PUML form.

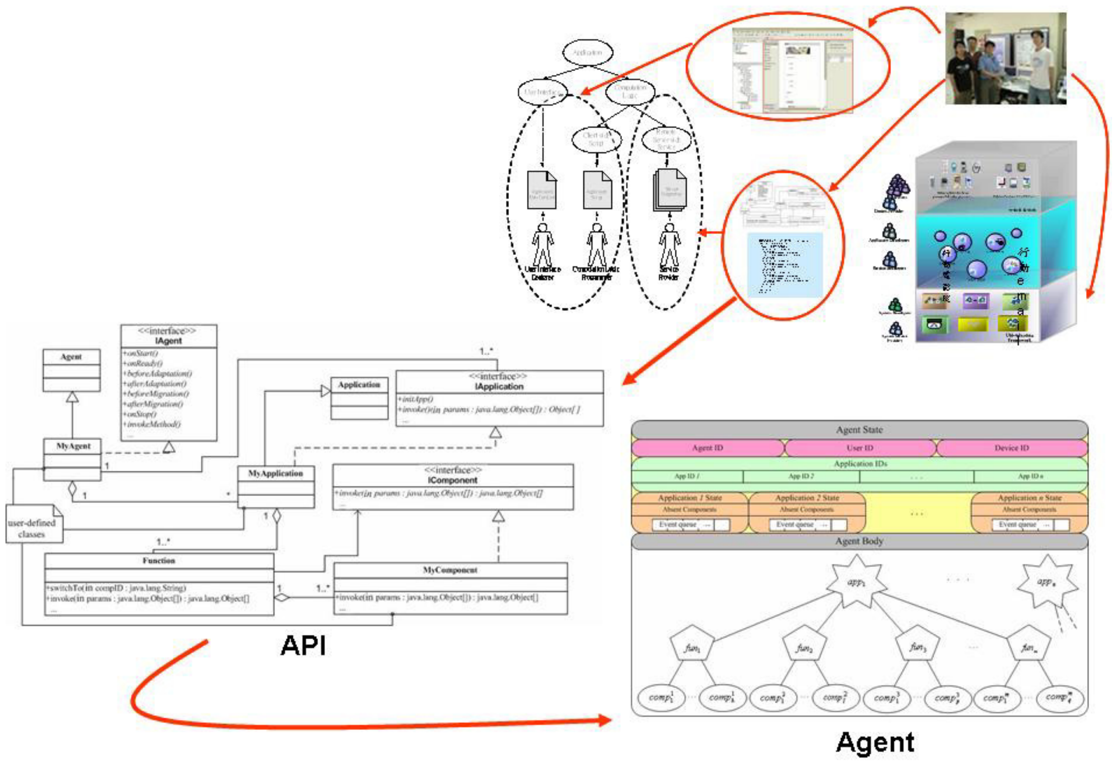


Fig. 42 the toolkit we designed

5.1.3 The generated code

The following code (Listing 15) is the WML code which generated from UIExample.puml. In the code, there are three `<input>` elements which are converted from the three `<textnote>` elements in Listing 14. The top three of Figure 43 demonstrate simulating the WML code generated. For example, a user inputs two number, 1 and 2, in Step 1, and selects action 2 in Step 2 subsequently. Then, the `<go>` would be performed. However, there is a problem—how to accomplish passing the values of the numbers into the WMLS function, and updating the `value` attribute of the `<input name="sum" … />` element—must be coped with. The trick we used is exploiting `WMLBrowser.getVal()` and `WMLBrowser.setVal()`. `WMLBrowser.getVal()` can be used to get the value of the variable specified from the WMLBrowser environment. Specifically, the values of the variables, standing for the two `<input>`, can be obtained by invoking `WMLBrowser.getVal()` in the WML Script. Returning the computed result can be completed through `WMLBrowser.setVal()`. addTwoNumBroker.wmls is the code generated in transformation to manipulate this event-handling. The related usage of the two methods can be referred in [40].

Listing 15. The WML code transformed from the PUML code in Listing 14

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
  "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>

  <card title="FirstPage" id="board1">
    …
  </card>

  <card title="SecondPage" id="board2">
    <p>Input two numbers:</p>
    <p>Number 1:</p>
```

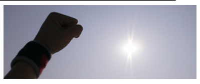
```
  <p><input name="num1" value="0"/></p>
  <p>Number 2:</p>
  <p><input name="num2" value="0"/></p>
  <p>Sum:</p>
  <p><input name="sum" value="0"/></p>
  <do name="action2" type="accept" label="action2">
   <go href="addTwoNumBroker.wmls#start('board2action2')"/>
  </do>
 </card>

</wml>
```

The code, shown in Listing 10, is transformed from the same PUML document (Listing 8). The `<puml:board name="">` … `</puml:board>` is converted to a `Brdboard2` class, which is the extended class inheriting the `Form` class of J2ME MIDP. In the code, the `TextField` control including `num1`, `num2`, `num3` are declared at the beginning, and initiated in the constructor of the `Brdboard2` class. They are transformed from the three `<puml:textnote>` elements in the original PUML document.

Respecting the event handling, the PUML code is transformed into that the `Brdboard2` class implements the `commandAction` method of the `CommandListener` interface. Once the method is invoked, namely some action was triggered, the code within the method would compare the name (ID) to see which object issues the event. If the action is triggered by `action2`, `sys_object1.sum(num1.getString(), num2.getString())` will be invoked to sum the two input numbers, shown in Line 51. In order to set the returned value back to the TextField `sum` in `Brdboard2`, we embed a method, `UIExample.sys_instance.getBoard("board2")`, in the generated code to obtain the board object specified, and `sum.setString()` to update the value of sum thereupon, shown in Line 49. The above code is transformed from the code in Line 28-33 in Listing 8. The tree figures, in the bottom of Figure 43, shows the simulating result. Similarly, an user inputs two numbers, and then chooses the action 2 to sum up the two number. Finally, the result is returned and displayed on the third TextField.

Listing 16. The J2ME code transformed from the PUML code in Listing 14

```
…
class Brdboard2 extends Form implements CommandListener {

  UIExample sys_ui;
  private addTwoNum sys_object1 = new addTwoNum();
  public StringItem mainTitle;
  public StringItem num1Title;
  public TextField num1;
  public StringItem num2Title;
  public TextField num2;
  public StringItem sumTitle;
  public TextField sum;
  private Command action2 =
     new Command("action2", Command.SCREEN, 1);

  public Brdboard2(UIExample sys_ui) {
    super("SecondPage");
    this.sys_ui = sys_ui;

    mainTitle = new StringItem("Input two numbers:", "");
    this.append(mainTitle);

    num1Title = new StringItem("Number 1:", "");
    this.append(num1Title);

    num1 = new TextField("", "0", 50, TextField.ANY);
    this.append(num1);

    num2Title = new StringItem("Number 2:", "");
    this.append(num2Title);

    num2 = new TextField("", "0", 50, TextField.ANY);
    this.append(num2);

    sumTitle = new StringItem("Sum:", "");
    this.append(sumTitle);

    sum = new TextField("", "0", 50, TextField.ANY);
    this.append(sum);

    this.setCommandListener(this);
    this.addCommand(action2);
  }

  public void commandAction(Command command,
    Displayable displayable){
     …
     if(command == action2){
       ((Brdboard2)UIExample.sys_instance.getBoard("board2")).
         sum.setString(
         sys_object1.sum(num1.getString(), num2.getString())
       );
     }
     …
  }
}
…
```



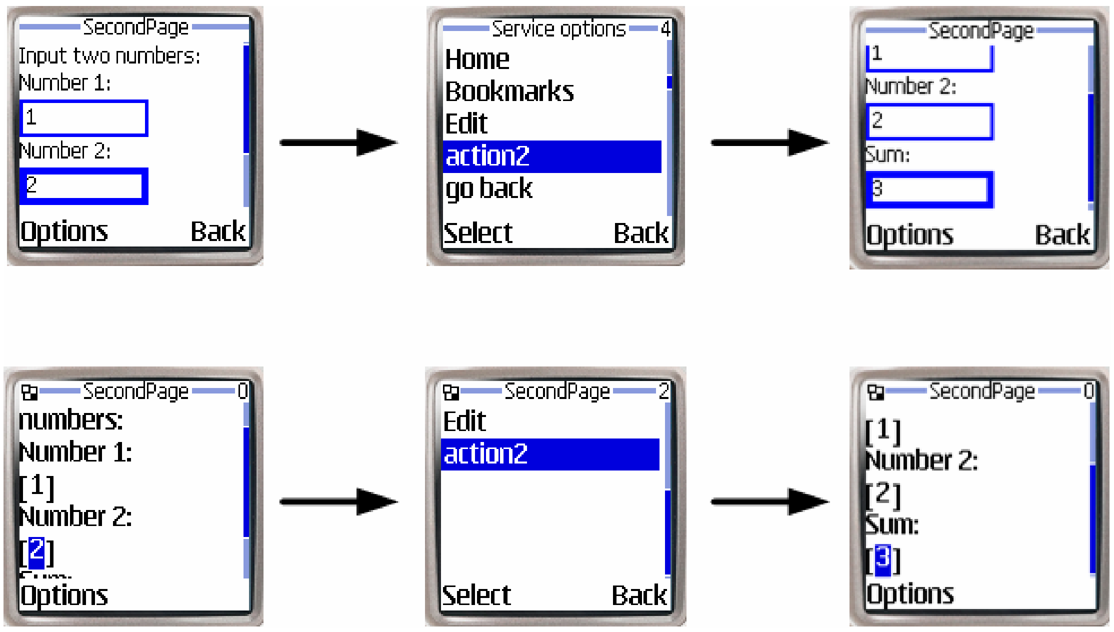


Fig. 43 The top figures are the simulating results of the WML code, and the bottom is the result of J2ME code generated from the code of PUML/PGML

5.2 Application component and agent development

In order to enable this framework to be aware of the structures of applications, we define *Application Structure and Component Constraints* (ASCC), an application profile description. Listing 17 illustrates the ASCC profile of the application ImageGathering.

Listing 17. The ASCC profile to describe structure of ImageGathering

```
<?xml version="1.0"?>
<ascc xmlns:ascc=http://dcsw3.cis.nctu.edu.tw/project/CAAS ...>
 <application id="ImageGathering">
   <function id="SubOrAdd">
     <default idref="Add"/>
     <component id="Add" priority="50%"
      stateful="No" relative="No" carried="No”>
     <component id="Sub" priority="50%"
      stateful="No" relative="No" carried="No”>
   </function>
 </application>
</ascc>
```

As we can see in Listing 17, the `<application>` element includes five `<function>` elements, which can describe the one function. In each `<function>`, the candidate components can be specified. Lines xxxx, for instance, declare that `<component id=″Add″ ...>`, and `<component id=″Sub″ ...>` can implement the `AddOrSub` function. In advance, within a `<component>` element, the properties, `stateful`, `relative`, and `carried`, can be used to set components stateful/stateless, relative/irrelative, and carried/un-carried respectively. The `priority` property concerns the priority of a component, one of which is chosen in each application adaptation. Furthermore, to set a component as a default component for a function, we can use the element `<default>`. If we want to set a component implementing the function which cannot be replaced with others, we can use the property “`unchanging=′Yes′`″. Figure 44 exhibits the class diagram of the implementation of the back-end module, which is made up of the classes derived from three original classes. A programmer defines a personal agent class, which is derived from the `Agent` class, and lets the agent carry the applications whose classes derived from the `Application` class. Furthermore, the programmer can define various subclasses of the class `Component` to substantiate and diversify his application. Without loss of generality, we use `MyAgent`, `MyApplication`, and `MyComponent` as the user-defined classes, which are illustrated in Figure 44.

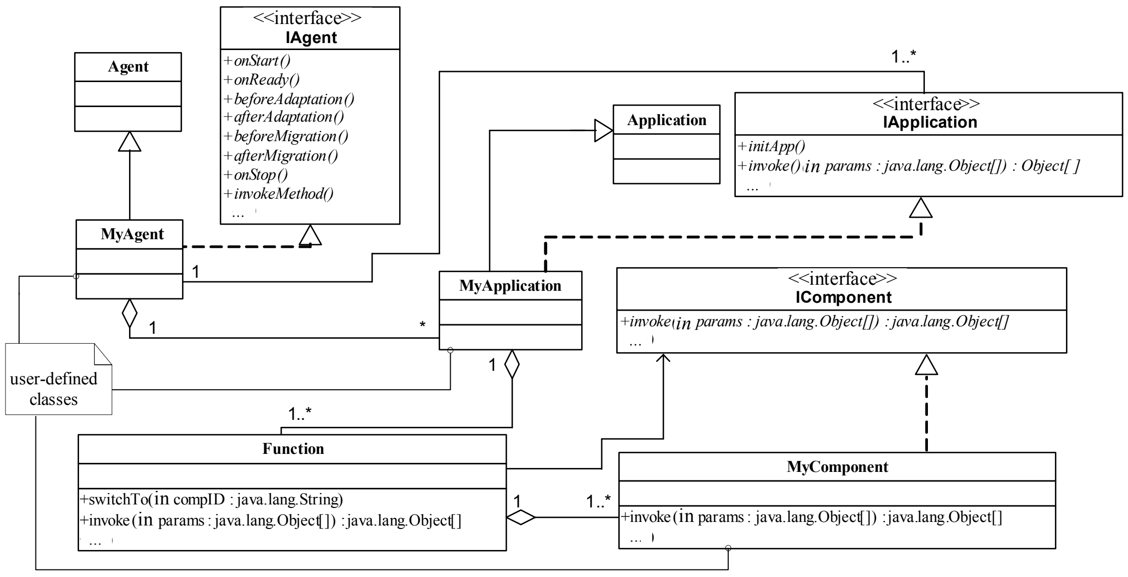


Fig. 44 The class diagram of programming agents and back-end modules

Chapter 6
Related work

In our design, remote dynamic invocation acts to complete invocations between the front-end and back-end modules. RMI, a method invocation on remote objects, is a widely used interaction paradigm. However, not all devices support RMI. Java reflection [41] (Section 4.4) lets programmers invoke the appointed method of the object determined dynamically at runtime. The mature RMI and Java reflection techniques enable programmers to develop applications efficiently, but have not been completely supported in mobile execution environments. For example, in the J2ME runtime environment, Sun Microsystems has not defined the RMI mechanism in the J2ME specification. Though Sun Microsystems defined the RMI interfaces on the CDC environment (an optional package of CDC), it did not provide the RMI interfaces on the CLDC environment. As can be seen in Figure 22, the devices being used in the CDC environment are PDA, Palm, Pocket PC, Smart Phone, while the devices with lower computational power only provide the CLDC environment.

Most mobile agent systems [42, 43] provide abundant functions, including agent migration, communication of agents with other agents and with the underlying system, as well as support for security, transactions and controlling agents. For instance, MOLE [44] offers an agent migration infrastructure with all of these functions, such as a protocol for fault-tolerant execution of mobile agents, accounting and billing, and control algorithms for finding agents, terminating agents, and orphan detection. Though complete functions support the mobile agent, adapting application according to the characteristics of the small and handheld devices has not been provided yet.

Some previous research has focused on the intrinsic structure of mobile agents and mobility behaviors of mobile agents, such as MobileSpaces [40]. MobileSpaces proposes agent hierarchy and inter-agent migration. The former is so that an agent can have several child agents, each of which also has agents as its child agents, and so on. The latter means that an agent is capable of migrating into another computer or to within an agent. Also, this framework makes agents adaptable. It regards a mobile agent as a component, and can combine a collection of agents into a single agent. Several agents are organized hierarchically into one agent. Additionally, this compound mobile agent can be adapted to the target environments. Although the hierarchical structure and adaptable concept for the mobile agents are provided in this framework, it does not structure the application and consider the context-aware adaptation for various mobile devices.

m-P@gent [45, 44] provides environment-aware mobile agents capable of running on resource-limited devices and appliances. In addition, it supports the runtime environment with mobile applications on the mobile devices, and contains four subsystems - @Desk for the PC platform, @Palm for the Palm device platform, @Pocket for the PocketPC platform, and @TINI for the TINI device platform. Moreover, it divides a mobile agent into two parts: a core and add-on functional modules. Then, it can adapt add-on modules of the agent to a runtime environment via a specific profile for each runtime environment, such as are profile for J2SE and another profile for J2ME. Yet this framework lacks the ability to distribute the computational loading of applications on the small and handheld devices. In other words, capabilities of the applications on this mobile agent system are restricted by the limitations of the devices. Furthermore, to adapt each component of the mobile agent, it is necessary to describe the type and class of a component for each runtime

environment. In our system, only description of component constraints in an ASCC profile is needed for the same purpose.

On the other hand, some researchers [46, 47, 48] have explored the follow-me applications. Harter et al. [48] describe a sensor-driven, or sentient, platform for context-aware computing that enables applications to follow users while they move around a building. Takashio et al. [47] also propose a mobile agent framework *f*-Desktop for the migration mechanisms of follow-me applications in an ubiquitous computing environment and evaluate its basic performance. Even though the basic functions of migration and adaptation of applications are provided, this framework does not concern the real context profiles of mobile devices for adaptation, and does not help run applications on these mobile and embedded devices.

In context sensing and modeling, Schmidt has explored context acquisition from sensors [49], and aim to model the context information [50, 51]. Gray et al. [50], present a way of analyzing sensed context information formulated to help in the generation, documentation and assessment of the designs of context-aware applications. Furthermore, to use CC/PP as the context information, Indulska et al. [52] address a context model and a context management system able to offer pervasive systems, and discuss the pros and cons of the CC/PP framework.

For developing context-aware applications, Dey et al. [53] describe a distributed software infrastructure to support context-aware applications in the Aware Home, a prototype smart environment. Their infrastructure is similar to the Situated Computing Service [54]. Both of them discuss polling and notification mechanisms to impart applications information of context changes. Kermarrec et al. [55] focus on a contextual object, a conceptual object model, for developing applications toward adaptation on the continuous changes of the mobile environment. A contextual object

has a context-sensitivity list (similar to component constraints in our framework) for describing the dependencies of an object and the kind of context that it senses. In addition, it has a reference to some real object (e.g. HTML page, Java Class, etc) to represent the value of this object in the current context. A conceptual framework for context-aware applications in current mobile and Internet environments has also been proposed [56]. The framework contains three parts. The first is the context management part capable of sensing and aggregating data, and managing the set of context groups. The second is the service management part that selects the appropriate services with context information from context management part, and returns the services to the adaptive user interface part. The third is the adaptive user interface part, which provides users with the adaptive and web-based user interface with selected services. All of the frameworks can facilitate the development of context-aware applications and a fundamental adaptation infrastructure for the applications on ubiquitous computing environment. Nevertheless, the weakness of their frameworks lies in the decision of the appropriate component or service for application adaptation according to context information.

Some agent systems are explained before. Besides, there are some systems with the same or similar functions. Table 11 briefly shows the related work. In the table, we classify the systems into the several types: Context Awareness (CA) [55][56], Framework Adaptation (FA) [57][58][59], Mobile Agent systems (MA) [43][42][60][61], Context aware agent (CMA) [62], Transformation Engine (TE) [63], Web Server (WS) [64], Context Awareness (CAA) [65]. Also, we list their functions: Form-based XML programming model, Mobile environment, Mobile Agent, etc. In the table, content adaptation means that content can be adapted by certain algorithms. An example is picture encoding. G^2 does not have this function, but can provide the

adaptation mechanism. In this mechanism, programmers can use some picture encoding algorithms to adapt pictures to different situations.

Tab. 11 the related work

	CA	**FA**	**MA**	**CMA**	**TE**	**WS**	**CAA**	**G^2**
Form-based XML programming model						**V**		**V**
Mobile Environment			**V**					**V**
Mobile Agent			**V**	**V**				**V**
Web-based	**V**					**V**		**V**
Content adaptation	**V**	**V**			**V**	**V**		
Context Awareness				**V**		**V**	**V**	**V**
Service adaptation		**V**						**V**

(**V** – has this function, e.g. CA is Web-based and has content adaptation function)

Chapter 7
Conclusion and Future Work

In summary, let we take CAAS as an example. We have explained our focus on transmitting agents efficiently and adapting applications to cope with the variability of user devices. By means of the front-end module and the back-end module, the restrictions of developing applications on small and mobile devices can be decreased. Furthermore, agents can synchronously migrate with their owners or be asynchronously anchored to their resident server. To transmit the agent efficiently, we experiment on agent migration strategies, and use the LAM as the default strategy for the agent migration. Additionally, by structuring applications in ASCC profiles, and leveraging CC/PP and WAP UAProf frameworks, the attribute-based component decision algorithm can choose the components suitable for the context of the user's devices.

Currently, there are some issues, including the replacement of the stateful and relative components, the conflict of the component property declaration, the consistency between the ASCC profile and the back-end module, and the lack of proper component declaration. Therefore, in the future we will attempt to design a software development kit (SDK) to aid programming and consistency checking. To further enhance this framework, some services related to the integration of this framework will be discussed in the future. Transaction, security, and server scalability handling, as well as load balancing and faulty recovery can be achieved by including services of distributed computing platforms, such as J2EE [73]. The J2EE

environment offers a distributed application model, a unified security model, flexible transaction control, etc. In transaction, several invocations between the front-end module and the back-end module of an application are regarded as an atomic unit. This transaction can be handled through some particular operations, such as commit or abort, and the two phases commit protocol. Security consists of authentication and authorization, which can be used to protect servers against malicious applications, and vice versa. Because some vendor's implementations of J2EE have the capability for scalability issue, we can use the J2EE framework to play the infrastructure for our system implementations.

In addition to the methodologies, we will attempt to integrate our framework with some mobile agent systems. IBM Aglet [42] and MOLE [43], for instance, have full-fledged mechanisms of security, transaction, scalability, etc. Furthermore, we intend to exploit the context sensing and modeling technologies to enlarge the use of contextual information toward adaptation in ubiquitous computing environment.

We can conclude our work in a generic model, shown as in Figure 45. The model contains six parts: Profiles, Performers. Sensors, Environment, Decision engine, Applications. In Chapter 4, we realize these functions in three systems. They contain context awareness, adaptation/transformation, agent migration, and application model. Besides the instances, we can add other techniques, such as Zig-Bee [66] as a sensor component, a rule engine (JESS [67]) as the decision engine in our newly-created system.

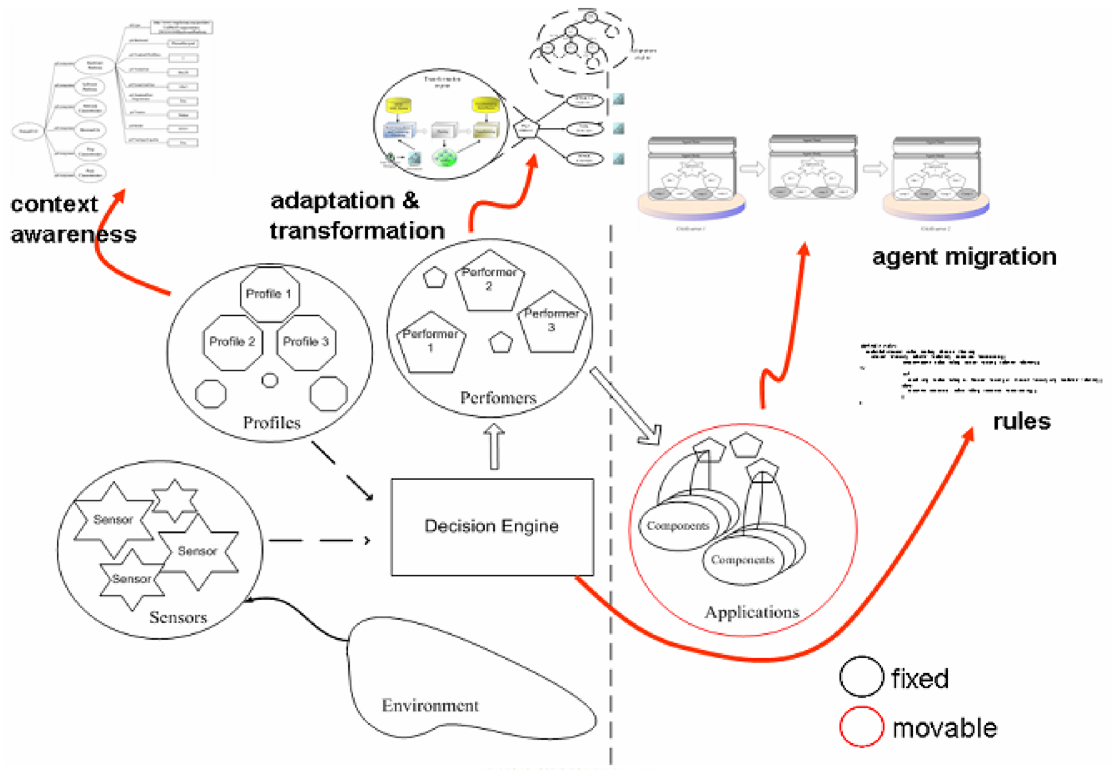


Fig. 45 a generic model

Figure 46 displays the overview of G^{2}MR which we are realizing in progress. In G^{2}MR, we use RFID for detecting users' location information and Java MHP (Multimedia Home Platform) [70] for displaying the result to users. Figure 47 illustrates the system architecture of G^{2}MR. In the system, we want to add Decision Maker, Event Manager, etc.

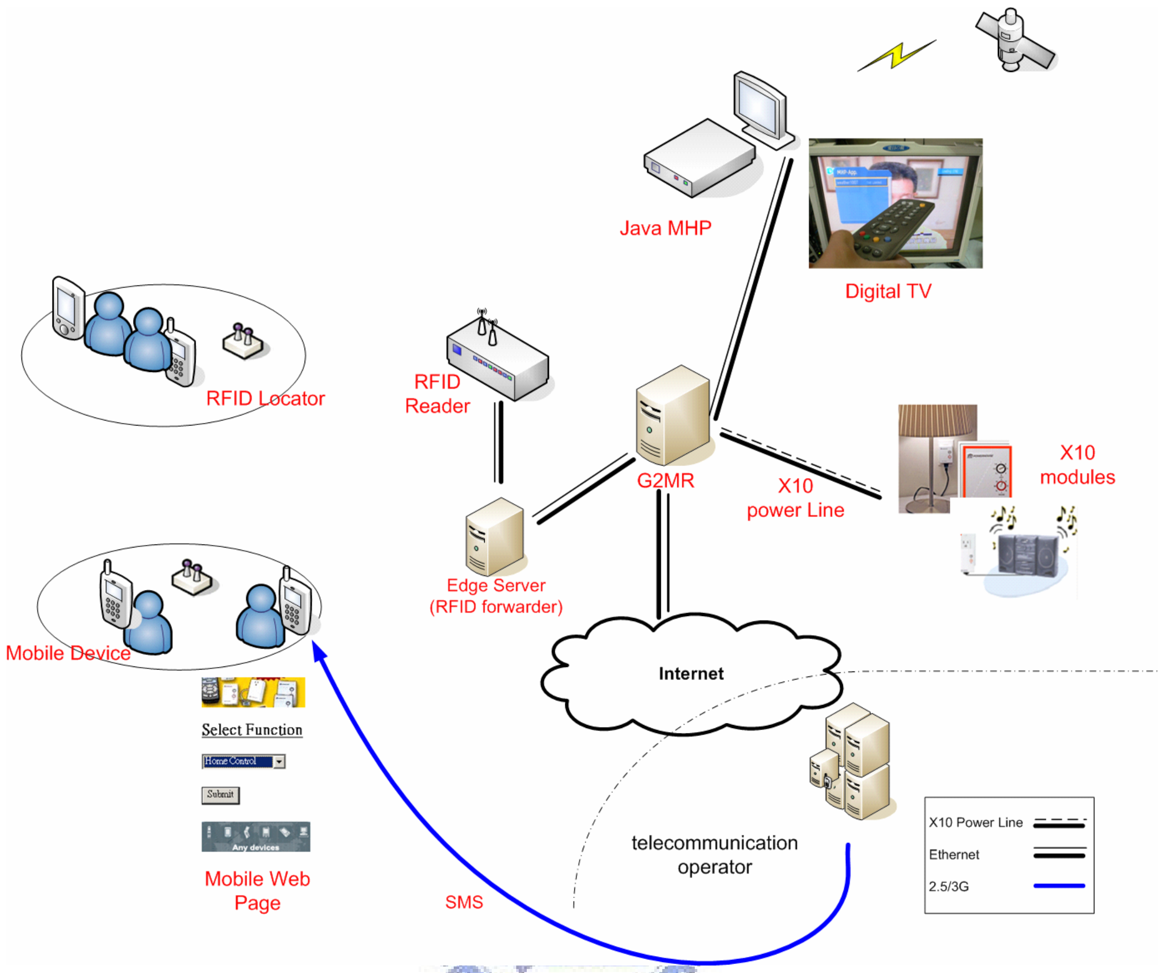


Fig. 46 an overview of G^2MR

b

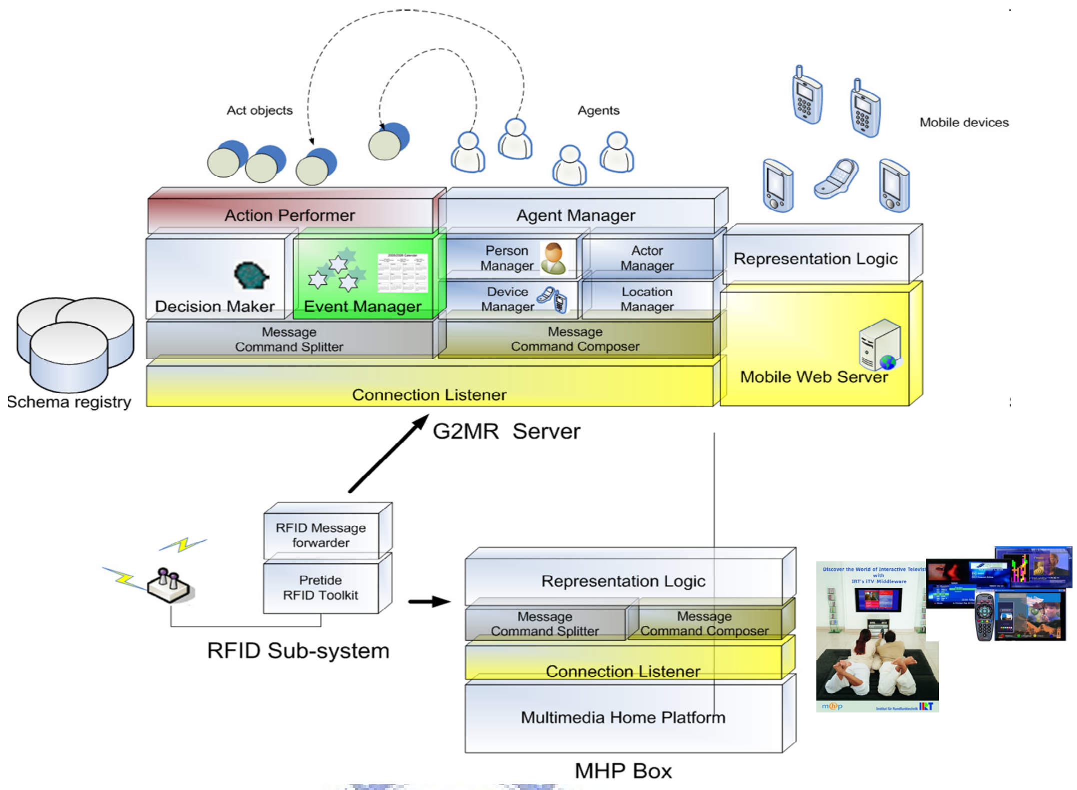


Fig. 47 the components of G^{2}MR

Table 12 illustrates the work we have implemented and the future work we want to design and implement. As you can see in this table we can see the functions provided by the systems and the revolution of these systems. There are some new emerging technologies: Web 2.0 [69], 3G [70], mobile streaming, DVB-H [71], etc. We can consider these technologies to enhance our system. For example, there is a technology called architecture of participation [72] in Web 2.0. Its means client device can have the detecting function to be aware of context of a user. There are a great many of applications applying these emerging discovered technologies for ubiquitous computing. We hope we can approach the goal to accelerate convenience for human lives.

Tab. 12 The system we implement and the future work

Name	Ubi-Adapting	G^2	G^2MR
Profile	CC/PP WAP UAProf	CC/PP WAP UAProf	N/A
Sensor	N/A	Bluetooth	**RFID**
Performer	PUML transformer	PUML transformer	XHTML MP Generator
Application	Application Migration	Application Migration	N/A
Decision Engine	N/A	N/A	**JSSE**
Embedded Box	No	Yes	Yes
Application Practice	Library, Movie, Home control	SMS, Schedule Home Control	**MHP**

Web 2.0
3G
Mobile Streaming
DVB-H
…

Aware Home
Aware Building
….

References

[1] Bill N. Schilit, Norman Adams, Roy Want, “Context-Aware Computing Applications,” *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, 1994; 85-90.

[2] Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, Paul Webster, “The Anatomy of a Context-Aware Application,” Mobile Computing and Networking, 1999; 59-68.

[3] Chen, G. & Kotz, D, “A Survey of Context-Aware Mobile Computing Research,” Technical Report, Dartmouth Computer Science Technical Report TR2000-381, Hanover, New Hampshire, November 2000.

[4] Cheverst, K., Davies, N., Mitchell, K., Friday, A. & Efstratiou, “Developing a Context-Aware Electronic Tourist Guide: Some Issues and Experiences,” Proceedings of the SIGCHI conference on Human factors in computing systems, 2000; 17-24.

[5] Asthana, A., Cravatts, M. & Krzyanowski, P, “An indoor wireless system for personalized shopping assistance,” Proceeding sof IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, 69-74. 1994.

[6] Kazunori Takashio, Gakuya Soeda, Hideyuki Tokuda, “A Mobile Agent Framework for Follow-Me Applications in Ubiquitous Computing Environment.,” Proceedings of 21st International Conference on Distributed Computing Systems Workshops (ICDCSW '01), Mesa, Arizona, 2001.

[7] WAP, http://www.wapforum.org/.

[8] Sun Microsystems, “Java 2Platform Micro Edition Technology for Creating Mobile Device,” Sun Microsystems, Inc, 2000.

[9] JSR 118 Expert Group, JSR-000118 Mobile Information Device Profile 2.0 (Final Release), May, 2002. http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html.

[10] PersonalJava, http://java.sun.com/products/personaljava/.

[11] Microsoft Mobile Web Forms, http://samples.gotdotnet.com/mobilequickstart/(mgk4rd2jnyo1zm55tgnot02p)/Default.aspx.

[12] NET Compact Framework, http://samples.gotdotnet.com/quickstart/compactframework/.

[13] 3GPP, “TS 22.057 V5.4.0. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Mobile Execution Environment (MExE); Service description, Stage 1 (Release 5)” 2002, http://www.3gpp.org.

[14] Tzu-Han Kao, Yung-Yu Chen, Tsung-Han Tsai, Hung-Jen Chou, Wei-Hsuan Lin, Shyan-Ming Yuan, “PUML and PGML: Device-independent UI and Logic Markup Languages on Small and Mobile Appliances”, Lecture Notes in Computer Science (LNCS) of Springer-Verlag, The 2005 IFIP International Conference on Embedded And Ubiquitous Computing (EUC-05), Nagasaki, Japan, 6-9 December 2005. (SCI)

[15] Tzu-Han Kao, Sheng-Po Shen, Shyan-Ming Yuan, and Po-Wen Cheng, “An XML-based Context-Aware Transformation Framework for Mobile Execution Environments,” Lecture Notes in Computer Science (LNCS) of Springer-Verlag (APWeb 2003, Xian, China), Vol. 2642 / 2003, pp. 132 - 143.

[16] Tzu-Han Kao and Shyan-Ming Yuan, “Automatic adaptation of mobile applications to different user devices using modular mobile agents,” Software:

Practice and Experience. Published Online: 27 Jun 2005 (SCI)

[17] Tzu-Han Kao, Yi-Hsiang Chou, Ming-Chun Cheng, Hsin-Ta Chiao, Shyan-Ming Yuan, "The design and Implementation of a Mobile Agent-Based Framework for Context-Aware Computing," Proceeding of ICS2002. International Computer Symposium (ICS2002). Dec. 18 - 21, 2002, Hualien, Taiwan.

[18] Tzu Han Kao and Shyan-Ming Yuan, Designing an XML-based context-aware transformation framework for mobile execution environments using CC/PP and XSL," Computer Standard & Interface. Available online 6 November 2003. (SCI)

[19] Ricardo Devis, "The Object-Oriented Page," June 1997. http://www.well.com/user/ritchie/oo.html

[20] WAP Forum, "Wireless Markup Language Specification Version 1.1," Jun 1999. http://www.wapforum.org/

[21] Sun Microsystems, "Java 2 Platform, Micro Edition, 1.0a," December 2000.

[22] J. Gosling, B. Joy, and G. Steele, The Java Language Specification. Addison-Wesley, September 1996. http://java.sun.com/docs/books/jls.

[23] Hiroshi Maruyama, Kent Tamura, Naohiko Uramoto, Makoto Murata, Andy Clark, et al., XML and Java Second Edition: Developing Web Applications, Addison-Wesley, 2002.

[24] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, et al., "Extensible Markup Language (XML) 1.0 (Third Edition)," W3C Proposed Edited Recommendation. October 2003. http://www.w3.org/TR/REC-xml.

[25] David C. Fallside, "XML Schema Part 0: Primer," W3C Recommendation. May 2001. http://www.w3.org/TR/xmlschema-0/.

[26] Mark H. Butler, "Implementing Content Negotiation using CC/PP and WAP UAProf," External Technical Report HPL-2001-190, 2001. http://www.hpl.hp.com /techreports/2001/HPL-2001-190.html.

[27] Hidetaka Ohto, Johan Hjelm, "CC/PP exchange protocol based on HTTP Extension Framework," W3C Note, June, 1999. http://www.w3.org/TR/NOTE-CCPPexchange.

[28] Dan Brickley, R.V. Guha, and Brian McBride, "RDF Vocabulary Description Language 1.0: RDF Schema", W3C Working Draft, January, 2003. http://www.w3.org/TR/rdf-schema/.

[29] Franklin Reynolds, Johan Hjelm, Spencer Dawkins, and Sandeep Singhal, "Composite Capability/Preference Profiles(CC/PP): A user side framework for content negotiation," W3C Note, 1999. http://www.w3.org/TR/NOTE-CCPP/.

[30] Ora Lassila, and Ralph R. Swick, "Resource Description Framework (RDF) Model and Syntax Specification," W3C Recommendation, February, 1999. http://www.w3.org/TR/REC-rdf-syntax.

[31] Hiroshi Maruyama, Kent Tamura, Naohiko Uramoto, Makoto Murata, Andy Clark, et al., XML and Java Second Edition: Developing Web Applications. Addison-Wesley, 2002.

[32] Ann Wollrath and Jim Waldo. "Trail: RMI, " http://java.sun.com/docs/books/tutorial/rmi/index.html.

[33] Mark H. Butler. "DELI: A DElivery context LIbrary for CC/PP and UAProf," External Technical Report HPL-2001-260, Feb., 2002. http://www.hpl.hp.com/personal/marbut/DeliUserGuideWEB.htm

[34] Brian McBride, Andy Seaborne, Jeremy Carroll, "Jena Tutorial for Release 1.4.0," April, 2002. http://www.hpl.hp.com/semweb/.

[35] Geier, Jim. Wireless LANs. SAMS, 2002.

[36] Held, Gilbert, Data over wireless networks: Bluetooth, WAP, and wireless LANs. McGraw-Hill, 2001.
[37] Wang, Jiangzhou, Broadband wireless communications: 3G, 4G, and Wireless LAN. Kluwer Academic Publishers, 2001.
[38] Borland JBuilder. http://www.borland.com/us/products/jbuilder/index.html .
[39] Microsoft Studio .NET. http://msdn.microsoft.com/vstudio/ .
[40] Ichiro Satoh, "MobileSpaces: A Framework for Building Adaptive Distributed Applications using a Hierachical Mobile Agent System," Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000). Taipei, Taiwan, 2000.
[41] Dale Green, "Trail: The Reflection API," http://java.sun.com/docs/books/tutorial/reflect/index.html.
[42] Danny Lange and Mitsuru Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison Wesley, 1998.
[43] J. Baumann, F. Hohl, K. Rothermel, M. Strasser and W. Theilmann, "MOLE: A mobile agent system," Software-Practice and Experience, 2002; 32:575-603.
[44] Kazunori Takashio, Masakazu Mori, Masataka Funayama, and Hideyuki Tokuda, "Constructing Environment-Aware Mobile Applications Adaptive to Small, Networked Appliances in Ubiquitous Computing Environment, Lecture Notes in Computer Science, vol. 2574. Springer: Berlin, 2002; 230 - 246.
[45] Kazunori Takashio, Masakazu MORI, Hideyuki Tokuda, "m-P@gent: A Framework of Environment-Aware Mobile Applications for Small, Networked Appliances," Proceedings of 2002 IEEE 4th International Workshop on Networked Appliances. Gaithersburg, MD, USA, 2001.
[46] Phil D. Gray and Daniel Salber, "Modelling and Using Sensed Context Information in the Design of Interactive Applications," Proceedings of 8th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'01). Toronto, 2001.
[47] Jadwiga Indulska, Ricky Robinson, Andry Rakotonirainy, Karen Henricksen, "Experiences in Using CC/PP in Context-Aware Systems," Proceedings of Mobile Data Management, 4th International Conference. MDM 2003 (Lecture Notes in Computer Science, vol. 2574), 2003.
[48] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy, "Modeling Context Information in Pervasive Compututing Systems," Proceedings of The First International Conference on Pervasive Computing. Pervasive 2002 (Lecture Notes i) Computer Science vol. 2414). Springer:Zurich, Switzerland, 2002; 169-180.
[49] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, and W. V. de Velde, "Advanced interaction in context," Proceedings of First International Symposium on Handheld and Ubiquitous Computing. Karlsruhe, Germany, September 1999; 89-101.
[50] A.K. Dey, D. Salber, G.D Abowd, "A Context-based Infrastructure for Smart Environments," Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE '99), 1999; 114-128.
[51] Hull, R., Neaves, P., and Bedford-Roberts, J, "Towards situated computing," Proceedings of International Symposium on Wearable Computers (1997) 146–153.
[52] Kermarrec, A.-M., Couderc, P., ans Banatre, M, "Introducing contextual objects in an adaptive framework for wide-area global computing," Proceedings of the 8th ACM SIGOPS European Workshop, September, 1998; 229-236.

[53] Sei-Ie Jang, Joong-Han Kim, and R.S. Ramakrishn, ”Framework for Building Mobile Context-Aware Applications,” Proceedings of the First International Conference on The Human Society and the Internet - Internet Related Socio-Economic Issues citation 2001, July 04 -06, 2001.
[54] Eric Armstrong, Jennifer Ball, Stephanie Bodoff et al. The J2EE. 1.4 Tutorial. Sun Microsystems, Inc. November 16, 2003.
[55] Anita W. Huang, Neel Sundaresan, “A Semantic Transcoding System to Adapt Web Services for Users with Disabilities,” Proceedings of the fourth international ACM conference on Assistive technologies,156 - 163, 2000
[56] Chieko Asakawa, Hironobu Takagi, “Annotation-Based Transcoding for Nonvisual Web Access,” Proceedings of the fourth international ACM conference on Assistive technologies, 172 - 179, 2000
[57] Erlend Stav, Svein Hallsteinsen, Jacqueline Floch, “FAMOUS: Framework for Adaptive Mobile and Ubiquitous Services,” Reconfiguration Workshop, 27.01.2005
[58] Maria-Teresa Segarra, Francoise Andre, “A Framework for Dynamic Adaptation in Wireless Environments,” Proceedings of 33rd International Conference Technology of Object-Oriented Languages (TOOLS 33), 2000.
[59] Vasian Cepa, Mira Mezini, “MobCon: A Generative Middleware Framework for Java Mobile Applications,” Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 9, 2005.
[60] Satoh I., “MobileSpaces: A framework for building adaptive distributed applications using a hierarchical mobile agent system,” Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000), Taipei, Taiwan, 2000.
[61] Takashio K, Mori M, Tokuda H., “m-P@gent: A framework of environment-aware mobile applications for small, networked, appliances,” Proceedings of the 2002 IEEE 4th International Workshop on Networked Appliances, Gaithersburg, MD, 2001.
[62] Burstein, F., Zaslavsky, A., Arora, “Context-aware mobile agents for decision-making support in healthcare emergency applications,” Proceedings of the Workshop on Contextual Modelling and Decision Support (CONTEXT'05), Paris, France, July 2005.
[63] XSLT transformation engine, http://www.topxml.com/xsl/tutorials/intro/default.asp
[64] IIS 6.0 asp .net, http://asp.net/default.aspx?tabid=1
[65] context-awareness, http://en.wikipedia.org/wiki/Context_awareness
[66] ZigBee, http://www.zigbee.org/en/index.asp
[67] JESS, http://www.jessrules.com/
[68] JESS, http://www.jessrules.com/
[69] web 2.0, http://0rz.net/8c0LA
[70] 3G, http://en.wikipedia.org/wiki/3G
[71] DVB-H, http://www.dvb.org/
[72] The Architecture of Participation in Web 2.0, http://0rz.net/671A0
[73] J2EE, http://java.sun.com/javaee/