COMMPC - Component Based Middleware for Pervasive Computing

R. Nagaraja†, and Dr. G. T. Raju††

†Associate Professor, Bangalore Institute of Technology, Bangalore, Karnataka, India
††Professor and Head Computer Science and Engineering, R N S Institute of Technology, Bangalore, Karnataka, India

Summary
Portable devices like laptops, palm tops, PDAs or mobile phones have become widespread. Similarly network functionality like GSM, Bluetooth or WLAN has become a standard. Nevertheless, not many applications take mobility into account. An application and its communication functions are tightly coupled and the applications assume that network behavior does not change during the application use. Here in this paper we propose and implement a new component based lightweight middleware based on Service Component Architecture (SCA) consisting of a small foot-print core layer and a modularized pluggable infrastructure. The SCA eases the reconfiguration of the components at runtime to support different communication mechanisms and service discovery protocols. Besides using SCA, new functionalities can be added to the middleware platform that can be provided by remote applications. The architecture presented in this paper is suitable for mobile devices and extensible to make use of abstractions to conquer heterogeneity in mobile devices. The prototype of the architecture is implemented using fractal tools and tested on Nokia mobile phones and laptops. A minimum configuration of the platform can be executed on embedded systems. Resource-rich execution environment are supported by the extensibility of the middleware. The resulting core component is about 250Kbytes in size.

Key words:
CDC Connected Device Configuration, CLDC Connected Limited Device Configuration, SCA Service Component Architecture, QRPC Queued Remote Procedure call, IE Intelligent Environment, IIOP Internet Inter-ORB Protocol, BT Blue tooth, AC access mode, AM adhoc mode.

1. Introduction

Some of the challenges the mobile devices have to face are adaptability to changes (limited sources, bandwidth, peers, network conditions, services) and constant change from one environment to another (different base stations, different domains, varying communication rate etc.). There is a need to define and develop infrastructure and associated services which can be accepted to a broad range of environments from Internet based applications, to local networks, to mobile applications on PDA's and smart phones, to embedded systems. This can benefit in two ways. First, it enables the easier deployment of mobile applications in different environments by taking advantage of common platform provided by adaptable architecture. Second, due to change of technology, redevelopment of middleware has to be done to take benefit of new technologies. The impact of this will be helpful for software vendors and service providers to adapt their products and services more rapidly to new and emerging technologies. In the architecture one or more of the required component services can be loaded and started by a system level entity. This plug and play approach obviates the need for all middleware components to be running on a low power device at all times. Customizable architecture can optimize energy as they can be pruned depending on the workload and devices can be in contact using the available service interfaces. In order to provide a dynamic architecture for resource constraint devices, we created a dynamic lightweight container offering enterprise level capabilities for such devices.

With this architecture we are able to dynamically assemble communication part of an application. These components can then later on be adapted at runtime as per situation requirements. For example, mobile devices are having number of network interfaces for communication like GPRS, Bluetooth, Wireless LAN or IrDA. With our infrastructure, an application can dynamically supported by best possible network interface according to the situation that prevails or demands. This infrastructure can build multi-hop, multi-network bridge across several devices. This is the result of combination of SCA and protocol independent communication, as we are able to do arbitrary runtime changes to adapt communication part of application with acceptable overhead.

The application adaptation for a particular network interface is done by invocation broker in association with adaptation manager [11] and transport plug-ins. The adaptation is based on device status and context changes. The adaptation layer takes care of different communications.

The paper is structured as follows: section 2 presents the challenges to be addressed in mobile device environment, section 3 the existing middleware capabilities are described, section 4 presents proposed new middleware architecture, section 5 implementation, performance are

Manuscript received September 5, 2011
Manuscript revised September 20, 2011

explained and section 6 presents the conclusion and future work.

2. Challenges to be addressed

The networking interface used by portable devices range from infrared communications to Wi-Fi communication. The device interfaces used also changes due to the mobility. A device may change the network due to non-availability, cost of network and power requirement of the network. Distributed applications in this scenario are structured as application objects and services interacting with each other. Services in turn use device capabilities or further services of local device or of a remote device. Hence, the context-aware application has to tackle the following challenges.

- The Mobility, Accessing Methods and Functioning: Portable devices exhibit different properties like sometimes stationary and sometimes moving as they are carried by users. The device complexity varies from simple to stand-alone systems. There will be large variation in resources, capabilities and communication patterns. Thus the mobile devices have variation in functioning, mobility and accessing methods.
- The heterogeneous protocol service discovery: The heterogeneity in protocols to discovery and access service is handled by accommodating multiprotocol service discovery with different communication paradigms like RPC, event based [10] [20], request-response, publish/subscribe etc. [4] [14].
- Programming interface, protocol support and device support: The programming interface has to take care of heterogeneity of device capability to support application portability. Uniform programming interface can be achieved by modelling the device capabilities as components. In order to enable communication with devices having difference capabilities a device specific communication model can be used, separating the communication model from application. In order to allow all devices to use the architecture a minimum functionality is implemented on the devices and this can be enhanced depending on the availability of the resources in the device. Some of the above mentioned facilities are supported by classical middleware for distributed application for resource rich platforms.

3. Existing Middleware Capabilities

3.1 Some of Non Component based Middleware

Device heterogeneity is handled in classical middleware like CORBA [15], Java RMI or DCOM to provide homogenous access to remote entities independent of operating system and platform. One of the objective of these middleware is to provide as much as functions possible for effective use of the available resources. This objective is not suitable for resource constraint devices. In addition these middleware assume stable networks.

Conventional middleware are designed with dynamic reconfiguration capability to adapt their behaviour to changing environment and application requirements. These middleware do not support different communication models and different protocols for incoming and outgoing messages. The Rover kit provides this functionality for its QRPC concept on top of different transport protocols but addresses disconnected accesses to an infrastructure and not spontaneous networking.

The UIC is based on a micro-kernel that can be dynamically extended to interact with different existing middleware solutions. Still, the used protocol is determined before the start of the interaction and cannot be switched between request and replay and abstractions are provided only for remote services.

Gaia [8] provides an infrastructure to spontaneously connect devices offering or using services registered in Gaia. Gaia recreates an intelligent environment in which the user mobile devices are integrated on-the-fly when entering the area. To integrate the existing system, like CORBA, interaction between application objects is done via Unified Object Bus, which is layer on top of these systems. As essential system services, such as discovery and lookup are provided by Gaia infrastructure. Mobile device cannot cooperate automatically without the infrastructure.

In contrast to this, our architecture aims at supporting the cooperation of nearby devices using temporarily available hardware and software capabilities of nearby devices, independent of the presence of the external infrastructure [26] [28]. An infrastructure, such as IE, may be included into a spontaneous network as temporarily available services.

3.2 Some of Component Based Middleware

Several middleware approaches have been proposed to deal with pervasive environments. The trend to build container suitable for small devices had already started in many projects. Many micro kernel based container like Merlin [22], JBoss [16] and Spring [23] are small in size but they still relay on service available in infrastructure network like JNDI in JBoss or heavy use of XML in Merlin and J2EE services in Spring.

Micro-kernel architecture design for small devices like OSGi [24] and FRACTAL based on SCA [1], are available from Open Service Gateway Infrastructure and major software vendors. FRACTAL has been selected because of language independence and support for resource constraint

devices like CDLC compatible devices.
ReMMoc [5] is a reflective middleware to support mobile application development and uses OpenCOM [9] as its underlining component technology. OpenCOM [9] is implemented in C++ and available for few platforms. Another thing about ReMMoc [5] is that it provides dynamic reconfiguration in terms of binding and service discovery protocols. In order to do that, all functionality associated with different communication mechanism and service discovery must be in the device. In our case, the functionality should only be loaded when it is required.
AdaptiveBPEL [3] is a policy driven middleware for the flexible composition of Web Services. This approach is more focused on the automatic service integration, according to the application needs, than the middleware adaptation.
Prim [25] is a software architecture which provides programming language-level constructs for implementing components, connectors, configurations and events. The middleware is assembled before deployment according to the required components. So far no runtime assembly can be performed as required in a dynamic reconfigurable middleware.
GaiaOS [8][13] is a component based meta-operating system, that run on top of existing systems, such as Windows2000, WindowsCE, and Solaris. It is used in a middleware infrastructure for active spaces. The system focuses on the management of active space resources and provides location, context and event services.
Jadabs [6] is a dynamic lightweight Platform for Ad-hoc infrastructure based on component based container and aspect programming. The platform gives access to both local applications using local infrastructure as well as other devices using distributed infrastructure. It is developed for CDC compatible devices. Our platform is based on language independent component framework [12] and can configure applications using context configuration to CLDC compatible devices also.

4. Design Rationale

4.1 Requirements

Fulfilling the following criteria is the main goal in designing the new middleware.

- Reduced Footprint: The infrastructure should be small enough to fit into small devices without affecting the working environment of the devices.
- Dynamic Adaptation: The adaptation must be dynamic, automatic without user integration. Functionalities should be added and removed as per the requirement without stopping the application.
- Flexible Architecture: The functionalities required for adaptation (components, services etc.,) can be provided by local repository, related connected devices or a remote repository.
- Platform Independency: The platform independent architecture using components can be implemented on resource rich as well as resource poor devices and provides flexible platform for implementation of vendor specific applications.

The application programming interface contains the uniform abstraction of services as well as device capabilities via interfaces. The requests are directed to local service handler or remote service handler by invocation broker. Invocation broker allows different patterns like request/response, event, synchronous, asynchronous etc., for communication. A core model containing minimal functionality of accepting and dispatching request called Invocation broker is designed as a core component and other services like inter-operability protocols as well as object life cycle management [18] [19] are added as plug-in components. The invocation broker receives an invocation from an application or from a remote device and if the request is for a service available locally, it invokes appropriate methods to service the request using local service handler (Fig. 1). Remote service handler, which receive an invocation or a reply to a previous invocation (also represented as an invocation) are submitted to invocation broker to dispatch to corresponding local device service or device capability. In case of an invocation request for a service not available in the local device, the invocation handler dispatches the invocation to the remote device through remote service handler using the device network interface which can be used to reach the device at that instant (Fig. 2).

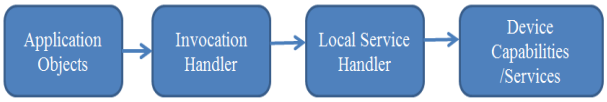

Fig 1: Local Device Capability / Service Access

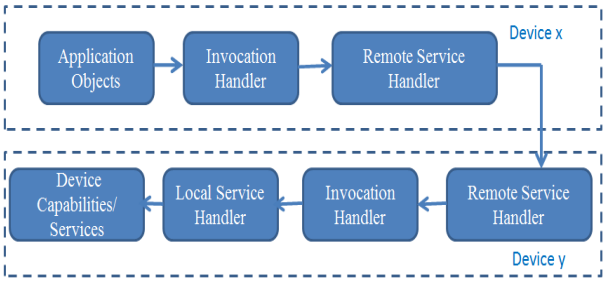

Fig 2: Remote Device Capability / Service Access

The remote service handler installed on the remote device receives and checks to see whether requesting service is available or not. If the service is available on that device the corresponding methods are invocated through local

service handler or the same procedure is used to dispatch request to a remote device.
The new middleware architecture is as shown in Fig. 3. The description of different modules is as follows:

4.2 Application Objects

Application objects generate method calls to invocation broker and uses context based conversion stubs for converting requests to invocation and skeletons are used for converting invocation to execute local services or device capabilities. In case of request (in the form of invocation) the invocation is dispatched to invocation broker for further processing. The invocation broker supports lookup services for local services and device capabilities.

4.3 Local Services and Remote Device Details

The local service registry maintains all locally available services and device capabilities. Services can be either application objects offering a service or device capabilities. Applications can query for available services by either specifying a name or the functional properties, i.e. the interface. The device registry maintains a list of all currently reachable devices and the transport plug-ins which provide the access to another device. A defined policy (like energy awareness, network interface) can be used to select a transport plug-in in case of availability of multiple transport plug-ins for a particular device. The underlying concept can be used to implement integration of other lookup mechanisms e.g. Jini and UPnP.

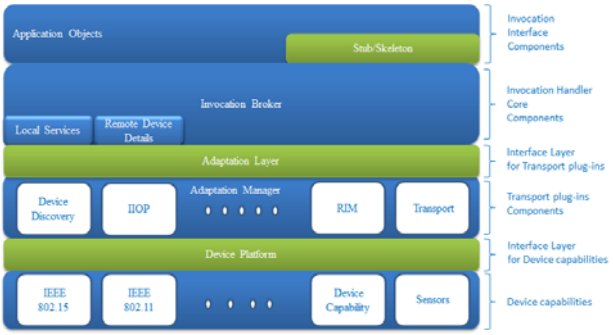

Fig 3: COMMPC Middleware Architecture

4.4 Invocation Broker

Invocation broker accepts invocation generated either manually or by a stub call. Invocation is also used by the invocation broker to access the registries for service lookups. An invocation is represented as object and is similar to dynamic invocation request in CORBA [15]. The format of the invocation is as shown in the Fig. 4.

Fig. 4 shows the elements of an invocation. Device and ServiceIDs are used to denote a sender and receiver of an invocation. ServiceIDs are unique and local to a device and this ID along with a unique DeviceID forms a unique ID globally. The message IDs are used for synchronization issues. A service context fields allows the specification of additional parameters that indicate the properties relevant to the processing of the invocation in the architecture such as synchronization issues or QOS parameters. Basically, the context is a name-value list where parameters are added freely. The payload contains the operations and parameters. In point-to-point communication the operations and parameters are interpreted as remote method invocation. In case of event based communication no receiver needs to be specified and the operation denotes the event-type on which application can subscribe [14] [17].

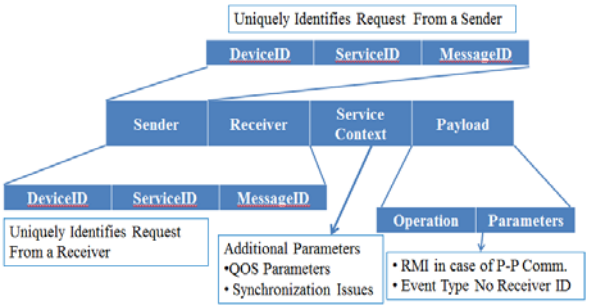

Fig 4: Invocation Object Structure

The invocation carries the target object and its messageID. If a MessageID is contained in the receiver field of the invocation, this indicates that a caller is either blocked or awaiting an asynchronous delivery of the invocation. In case of a blocked call the waiting thread is freed and the invocation is provided as return. In the asynchronous case the invocation broker uses remote invocation handler to call up the application through a callback. In this case the MessageID is used to designate the application callback registered at the invocation broker. Because of decoupling of communication model from the underlying interoperability protocols, a request/response based communication model can be realized over two event-protocols. An event can be sent as a single request in an RPC-based interoperability protocol. An interaction can take place over different transport plug-ins for out-going and in-coming invocations.

4.5 Adaptation Manager

The adaptation manager decides about loading required functionalities as per the situation identified by invocation broker through device and its network interfaces. The invocation broker on detection of a remote invocation gets device interface from device details table and forwards the

invocation to adaptation manager through Remote service handler. Remote service handler initiates appropriate calls to services of adaptation manager to implement different communication patterns like synchronous and asynchronous communication with the remote devices. Adaptation manager uses device interface given by invocation broker to decide the components to be loaded. The components can be loaded either from local storage or from remote repository. The adaptation layer can load the transport plug-in components as per parameters passed by invocation broker. At this level the capabilities of middleware can be extended as per the requirement to provide different level of services by loading appropriate transport plug-ins.

Platform specific capabilities, e.g. device capabilities and transports are represented as plug-ins (Implemented as components) and become accessible to the application programmers as services. Adaptation manager allows the dynamic loading and integration of new plug-ins.

Adaptation layer is responsible for accepting an invocation, marshal it, and transmit it as a protocol data unit to a remote peer, which then constructs an invocation by unmarshalling it. The simplest transport plug-in would use object serialization to marshal an invocation into a byte buffer and send the buffer via a transport protocol e.g. TCP/IP. Other transport plug-ins could rely on existing interoperability protocols and marshal and represent the invocation accordingly e.g., map it to a request-message in IIOP and marshal the parameter by CDR, which allows interoperability with CORBA based systems. The invocation broker may use any transport plug-in unless an application specifies a distinct transport protocol. The remote device details registry maintains a list of all currently available transport plug-ins to a specific device. Hence communication can take place as long as at least one transport plug-in allows the communication.

4.6 Device Platform

The device capabilities are provided in this layer and are represented as interfaces to be used by either transport plug-in components or by invocation broker (local service handler component). Required device capabilities can be loaded on demand in case of resource poor devices to save space.

5. Implementation and Performance

5.1 Component Based Prototype Design

A prototype of component based implementation (Fig.5) is designed to verify and test the performance of our concept of protocol independent communication among heterogeneous devices. The core component is composite component having Invocation Broker, Adaption Manager and Discovery Manager. Invocation Broker is implemented as a composite component with Invocation Handler component, Local Service Handler component and Remote Service Handler component. The local services and device capabilities are implemented in the Local Service Handler component as interfaces. Remote Service Handler component was implemented to handle the remote invocation call and incoming invocation from Adaptation Manager. The Adaptation Manager handles remote invocation as well as incoming invocations. This component dynamically loads the required transport components as per the device interface provided by Invocation Handler.

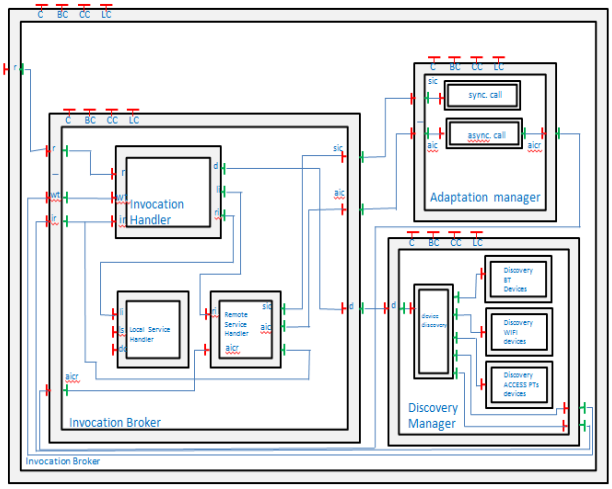

Fig 5 Component Architecture of Middleware

Discovery Manager component detects all devices in the neighborhood using device discovery protocols. These protocols are implemented as components and loaded whenever required by discovery manager for device discovery. Discovery Manager sends the details of devices found to Invocation Handler to record in device details table.

A prototype of the component based architecture is implemented (Fig.6) using SCA implementation Fractal [12] for components and ADL for configuration specification. Eclipse, maven and Fractal as plugin were used to develop platform specific codes. The middleware COMMPC is built on JULIA [2], a java implementation of the fractal component model. COMMPC core provides FRACTAL components with abstractions and tools to handle invocations and activities. COMMPC ADL extends JULIA's architecture description language with features for distributed deployment and dynamic application configuration. COMMPC library provides a library of components implementing functions like communication channels. COMMPC is built on top of these three modules. COMMPC is a message (invocation) based communication framework for network interface

transparent communication. The Invocation Broker with service and device registry, Adaptation Manager, transport plug-ins like TCP-transport, UDP-Transport and BT-Transport are implemented. The Invocation Broker handles different synchronization concept and the service context is used to indicate the synchronization of RPC calls. For synchronization invocation, sub and skeleton support is implemented as part of Invocation Broker.

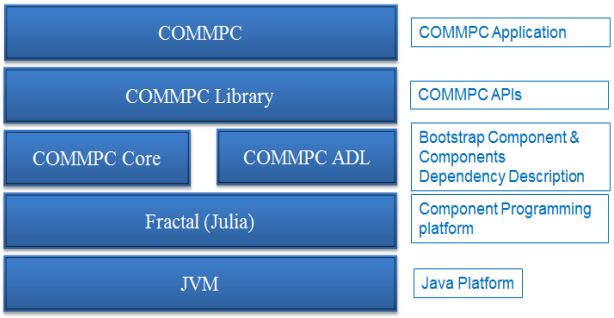

Fig 6: COMMPC Implementation Framework

5.2 Performance

The prototype is tested on two computing environments, one on laptop environment and another on mobile devices.
(i) Desktop Environment: The Three transport plug-ins are realized on laptop environment, two based on the java standard serialization mechanism on top of TCP/IP using laptops in adhoc mode and access point mode and a second based on Java RMI. The Adaptation Manager is implemented and allows the dynamic and static configuration of the system. The core model of the frame work occupied 250Kbytes as measured during the idle mode using task manager. During runtime, when invocation is exchanged, the system uses up to 600Kbytes. To measure the execution performance overhead introduced by the additional communication via the architecture, comparison of sending invocation via a Java RMI transport plug-in with a pure Java RMI-based system is done. The measurements were conducted for a synchronous RPC communication by transmitting invocations for an operation, which takes a single string input parameter and returns immediately. The strings were of different sizes. This was done for both local and remote invocations.
In case of local invocation (Fig. 7.), our method was clearly faster than RMI. This is due to the fact that RMI in this case uses the loop-back interface including the RMI and TCP protocol stack where as our core model forwards the call directly to the service skeleton and does not use the RMI based transport plug-in at all.
In case of remote invocation (Fig. 8.), the frame work introduces an additional performance overhead of about 20%. This is due to the fact of creating invocations from the stub objects and their interpretation by the skeletons.
(ii) Mobile Device Environment: Two of the protocols switching were implemented on mobile devices using Bluetooth J2ME optional package and Wireless Messaging API (WMA API). Jar files consisting of device discovery module, service discovery module, communication module and invocation broker were created and tested using Java MESDK3.0 and Emulator default CLDC phones. It was tested on Nokia mobile devices Nokia-3120, Nokia-7210 and Nokia-Express. We used WMA API 2.0 JSR 120 for GSM [27] and JSR 82 Release 3, JCP version 2.1 [27] for Bluetooth specification.

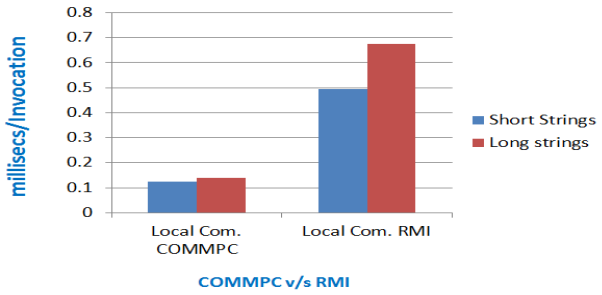

Fig 7: Local Communication Performance for both methods

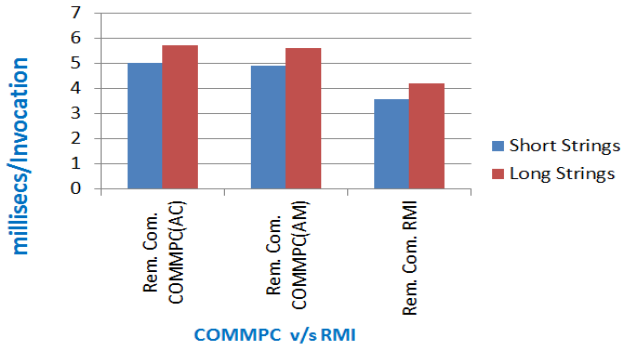

Fig 8: Remote Communication performance for both methods using different transport plug-ins

Table 1: Performance Details

Sl. No	Activity	Time Taken
1	Bluetooth Device Search Time	4.15 Sec
2	Communication Time using Bluetooth	Very less in the range of micro sec,
3	Communication Time using GSM	Less than 85 milli sec

The minimum required characteristics of mobile devices were 512KB of total memory and compliant implementation of J2ME CLDC. Table 1 shows the performance of prototype on mobile devices for transfer of an invocation using two communication channels namely Bluetooth and GSM.

6. Conclusion and future work

In this paper, we have introduced component based platform for mobile devices. With this platform, users can access different resources like services on different devices. The platform also supports multiple communication protocols and models. We have chosen Fractal as an approach to build our platform to take advantage of SCA and component paradigms. In this paper we showed how to use platform to access local as well as remote services using multi-protocol communication as well as multimode communication. We tested our platform using CDLC compatible mobile phones platform and laptops.

Table 2: Comparison of Microkernel Middleware with COMMPC

Features	COMMPC	ReMMoc	Jadabs	Adaptive BPEL	Prim
Platform independent	Yes	No	No	No	No
CLDC compatible	Yes	Yes	No	No	No
Middleware adaptation	Yes	No	No	No	No
Resource location	Local/ Remote	Local	Local	Local	Local
Dynamic adaptation	Yes	yes	yes	Yes	No
Multiprotocol in single session	Yes	No	No	No	No

A Comparison of features offered by the existing middleware and current middleware for pervasive computing (COMMPC) is presented in Table 2.

Currently, we are in the process of extending the functionalities to include other limited resources like memory and bandwidth for adaptation. To show the feasibility of our approach for accessing web services by mobile devices we are working on extending the functionalities of application, considering the available limited resources using model driven approach [7].

References

[1] Service Component Architecture specifications, http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications, 2007.

[2] Eric Bruneton, Thierry coupaye, Maltheu Leclercq, Vivien Quema, and Jean-Bernard Stefani, “An Open component Model and its support in Java”. In proceedings of the International Symposium on Component based Software Engineering (CBSE 2004), Edinburgh, Scotland, 2004.

[3] A. Erradi and P. Maheshwari, “Adaptivebpel: Policy-driven middleware for flexible web services composition,” In MWS 2005 Workshop at EDOC 2005, pages 5–12, Enschede, The Netherlands, 2005.

[4] C. A. Flores-Cort´es, G. S. Blair, and P. Grace, “A multiprotocol framework for ad-hoc service discovery,” In MPAC ’06: Proceedings of the 4th international workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC 2006), page 10, New York, NY, USA, 2006, ACM.

[5] P. Grace, G. S. Blair, and S. Samuel, “Remmoc: A reflective middleware to support mobile client interoperability in CoopIS/DOA/ODBASE,” pages 1170–1187, 2003.

[6] Jadabs, “Dynamic Lightweight Infrastructure for Small Devices,” http://jadabs.berlios.de.

[7] C.Parra and L. Duchien, “Model-driven adaptation of ubiquitous applications,” In 1st International Workshop on Contextaware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS 08), pages 97–102, Oslo, Norway, June 2008.

[8] R. Cerqueira, C. K. Hess, M. Rom´an, and R. H. Campbell, “Gaia: A Development Infrastructure for Active Spaces,” In Workshop on Application Models and Programming Tools for Ubiquitous Computing (held in conjunction with the UBICOMP 2001), Sept. 2001

[9] M. Clarke, G. Blair, G. Coulson, and N. Parlavantzas, “An Efficient Component Model for the Construction of Adaptive Middleware,” Proceedings of Middleware 2001, Nov. 2001.

[10] G. Cugola, E. Di Nitto and A. Fuggetta,“The JEDI eventbased infrastructure and its application to the development of the OPSS WFMS,” IEEE Transactions on Software Engineering, 27(9), 2001.

[11] Daniel Romero, Carlos Parra, Lionel Seinturier and Laurence Duchien, Rubby Casallas, “An SCA-Based Middleware Platform for Mobile Device,” EDOC Conference (2008), DOI : 10.1109/EDOCW.2008.17.

[12] Fractal, ObjectWeb, Open Source Middleware, 2004, http://fractal.objectweb.org/.

[13] M. Roman, and R.H. Campbell, “GAIA: Enabling Active Spaces “, Proceedings of the 9th ACM SIGOPS European Workshop, pp. 229-234, Kolding, Denmark, September 2000.

[14] A. Frei, A. Popovici, and G. Alonso, “Eventizing Applications in an Adaptive Middleware Platform,” Technical Report TR 451, Swiss Federal Institute of Techonolgy Zurich, Mar. 2004.

[15] OMG, CORBA Messaging, report orbos/98-05-06, 1998.

[16] J. Group. Jboss. http://www.jboss.org.

[17] M. Haupt, M. Mezini, M. Cilia, and A. P. Buchmann, “Towards Event-Based Aspect-Oriented Runtime Environments, Technical Report TUD-ST-2002-01,” Software Technology Group, Darmstadt University of Technology, Alexanderstrasse 10, 64289 Darmstadt, Germany, 2002.

[18] H.Cervantes and Richard S. Hall, “Automating Service Dependency Management in a Service-Oriented Component Model,” In Proceedings of the 6th Workshop on

Component-Based Software Engineering (CBSE), May 2003.
[19] IBM, Service Management Framework, 2004. http://www-306.ibm.com/software/wireless/smf/.
[20] Y. D. Bromberg and V. Issarny, "Service discovery protocol interoperability in the mobile environment," In T. Gschwind and C. Mascolo, editors, SEM, volume 3437 of Lecture Notes in Computer Science, pages 64–77. Springer, 2004.
[21] C. Szyperski, "Component Software: Beyond Object-Oriented Programming," Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
[22] Merlin.The Apache Avalon Project, 2004. http://avalon.apache.org/merlin.
[23] Spring. Java/J2EE Application Framework 2004, http://www.springframework.org.
[24] OSGi, Open Service Gateway Initiative. OSGi Service-Platform, IOS Press, release 3 edition, Mar. 2003.
[25] M. Mikic-Rakic and N. Medvidovic, "Adaptable Architectural Middleware for Programming-in-the-Small-and-Many," In Proceedings of the 4th ACM/IFIP/USENIX International Middleware Conference, pages 455–473. Springer-Verlag, June 2003.
[26] R. Nagaraja, Dr G. T. Raju, "Multiprotocol Communication for Mobile Devices," IET-International Conference on Next Generation Networks-2010, page 68-73, Mumbai, India, 2010.
[27] Bluetooth details, midlet details, WMA developers.sun.com and java.sun.com
[28] R. Nagaraja, Dr. G. T. Raju, "SCA based Multiprotocol Communication for mobile devices", ISBN: 978-1-4244-7923-8, iNSPEC Accession Number: 11973847, Digital object Identifier: 10.1109/ ICETECT.2011.5760263, Dated 02th May 2011.

R. Nagaraja received the B.E., M. E., and MS degrees from Bangalore Univ. in 1985, 1989, and from BITS 1998, respectively. He is presently working as associate professor in Information Science and Engineering department at Bangalore Institute of Technology, Bangalore. His research interest includes building adhoc infrastructure for mobile devices, building self-adaptive systems, protocol independent communication, component based design for heterogeneous devices and model driven engineering. He is a member of IET, ISTE and IACSIT.

Dr. G. T. Raju received the B.E., M.E., and PhD from Bangalore University in 1992, 1995 and 2008 from Visvesvaraya Technological University. His is currently working as Professor and Head Computer Science and Engineering Department at RNSIT, Bangalore. His research interest includes data mining, artificial intelligence and neural networks. He is a member of CSI and ISTE.