

Mobile Application Architectures

Pasi Alatalo¹, Jarkko Järvenoja¹, Jari Karvonen², Ari Keronen¹, and Pasi Kuvaja¹

¹ Department of Information Processing Science, University of Oulu
P.O.Box3000, FIN-90014 OULU, Finland

² Department of Economics, University of Oulu
P.O.Box4600, FIN-90014 OULU, Finland

{Pasi.Alatalo, Jarkko.Järvenoja, Jari.T.Karvonen,
Ari.Keronen, Pasi.Kuvaja}@oulu.fi

Abstract. Modern computer architectures are large containing thousands of lines of code with complex and often distributed forms. Mobile and multi-platform solutions even increase this complexity. Modeling and managing a software architecture is important way of handling these complex systems, describing it to different actors inside software business and for analyzing and improving system performance. In this paper we go through mobile architectural structures and analysis of these with empirical mobile application development. We used different architectural views for analyzing mobile architectures and architecture role in development. The architecture and architectures role on the development has been studied in mobile application- and multi-platform service development.

1 Introduction

Telephone, mobile and fixed-line, has connected people for a long time. At mid of 90's the GSM (Global System for Mobile) provided mobile phones with short message (SMS) based communication channel. Data started to be part of the communication. The development in wireless communications generated many services that were based on data transactions. Valuable information services as well as entertainment services became part of mobile communication.

The market of wireless communication is developing. Data transactions enabling technologies, GSM and SMS got new properties and requirements with the introduction of Wireless application protocol (WAP). Wap facilitated online-connection between two terminals possible and converged Internet into the mobile communication.[2] Next steps on the development route were to integrate digital television (already partly functioning), third generation mobile phones and local networks working together.

This integration created a new problem called entity management. The main question is how to manage application development in certain platform so that it would work also in multi-channel environment? One possible answer is reasonable architecture that allows flexibility in development. Keeping attention on the

architecture during the development helps in focusing on the right tasks at the right time. In this paper mobile application development is considered from the product idea to the final product from the architecture development point of view. The paper is based on experimental research performed in MONICA and MOOSE-projects.

2 Software Architectures in General

Software architecture describes and demonstrates the content of created software. The form of software descriptions varies. Diverse need of information and requirements of architecture have caused variety in software architecture definitions. One definition of the software architecture is to describe the relationship between static and dynamic parts of the software. In other words it alludes in two important characteristics of the computer program: the hierarchical structure of procedural components and the structure of data. [5]

In the 90's software architectures have increased to one of the most significant parts of software engineering. Different technologies that are handled especially from software architecture point were spreading to industrial use. In addition there are continually developed more complicated applications that include thousands of lines of code and that are complicated to maintain and to reuse. Those are the real-world problems and demand that caused the emergence of different software architectures and will develop in the area of changing technologies and system requirements. In that way software architecture definition can nowadays be seen as a critical part of the software application development.

Borderline between architecture and structure of lowest part of software is thin, which may affect overlapping in software architecture design. One definition could be that software architectures include those features of software that do not change during the software process. [5]

2.1 Purpose of the Software Architecture

Modeling and documenting software architecture is important for many reasons. Stakeholders can look at structural features of software with the help of modeling and documentation. Using architectural model developers can analyze software at very early phase of the software process. Architecture is the first step in designing software itself and it defines stable ground for software development. Different architecture models can be standardized and named, and can be re-used in many subsequent applications. This leads to identification and documentation of general architecture models that are application independent. The architecture models are called architectural styles. Examples for architectural styles are layered architecture and repository architecture. [4][5]

2.2 Product Line Architecture

The set of software programs, which have similar structure and functionality, are called software product families. The software architecture, which is common to

software product family, is often called product line architecture. Layers, components and frames have significant role in product line architecture. The use of product line architecture is continuously increasing in software industry. In specific the increasing use of the product line architecture is motivated by the fact that they make reuse of software modules easier, that leads further into more reliable software programs, better time controlled projects, increased productivity of the development, less development risks, easier prototyping etc. [10]

2.3 Layered Software Architecture

Layered software architecture means software structure that consists of layers that are logically similar on different abstraction levels. The function of layered software architecture is that higher layer can use services, which lower layer produces. Normally that leads into a situation where the lower layers produce the most common services. On the other hand this means that the higher level the layer is described to the more independent the layer is from the application itself.

Layered software architecture can be described in three-layered architectural model, which consists of application layer, middleware layer and platform layer. Middleware offers platform independent common services like support for user interface. Platform layer offers for example operating system, communication software and other hardware dependent software (drivers etc.). See Figure 1.

Most common structure of layered software architecture is hierarchical and bypassing forbidding. Hierarchy means that direction of service request is always from higher level to lower level or service request is sent to same level. To forbid bypassing means that service request do not bypass next layer. Structures that allow bypassing are possible but are not so common. See Figure 2.

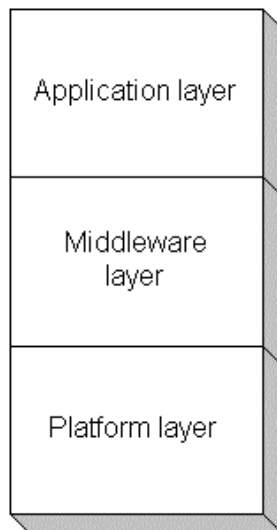


Fig. 1. Three-layer architectural model

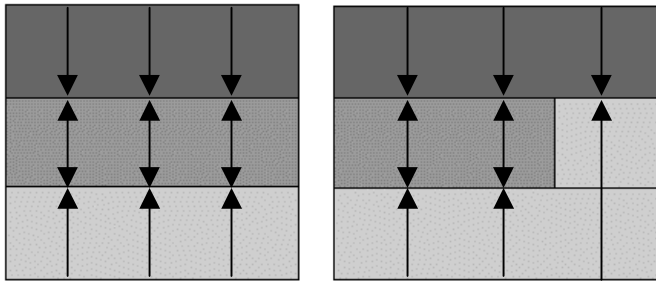


Fig. 2. Non passing and passing hierarchical layered software architectures

Independent layer needs interfaces to serve and request service for. Services are produced to serve higher layer requests that could be used via the interface. This interface is called service interface or utilization interface. On the other hand services, which layer is requested, could be used via implementation interface. Utilization interface of layer is the same as implementation layer of the higher layer. Implementation interface of the lowest layer and utilization layer of the highest layer connects system to physical environment.

Benefits of layered software architecture are clear, simple structure, which make possible to develop and understand software as sets of independent layers. Because of the independent nature of the architectures the designers who are specialized in different layers can focus onto their own layers without knowing anything about another layer. Additionally, layered software architecture makes understanding of system easier to them who are not so familiar with software designing and implementation for example customers and managers. [4][5][9]

2.4 Distributed Systems

Nowadays more and more applications work in distributed environments either in wired or wireless systems. Due to this, new distributed technologies are created and architectural solutions based on these new technologies are developed. The most significant of these technologies are CORBA (Common Object Request Broker Architecture) and Java RMI (Remote Method Invocation). CORBA is common distribution standard for objects and it is independent from implementation language and operating systems. Java RMI is Java implementation for distribution mechanism of objects. [11][12]

3 Different Views of Software Architectures

Different software architecture models and their representations have been used for different purposes. When analyzing more precisely the architectures and their representations, one dimension is the abstraction level. Higher abstract levels are used in general description of the software or parts of the software. The abstraction levels of the representations allow describing the basic functionality and structure of the software level by level and that makes it easier and faster. The more detailed

architecture descriptions we go the more complicated the architecture show-ups are. This also increases the requirements to understand the given information. The view that is under study is dependent of the stakeholder in question, such as end-user, client, project leader, implementer, manager or salesman, whose interests in software vary quite much. This report used on the mobile application development from the early idea until final product. The chosen views for architecture are based on the software architecture models presented by Philippe Kruchten. Kruchten defined so called „4+1 model“ to describe five different viewpoints for software. The viewpoints are illustrated here in graphical form in Figure 3. Here the Kruchten’s models will be applied in analyzing mobile applications and mobile application development. On each level of development the analysis will be performed through Kruchten’s viewpoints in order to find out conveniences of the mobile application development. [6]

The five viewpoints are as follow:

- **First view** is the use case view where the system is looked at from outside, as the user looks at it. This view emphasizes outer functionality of the system.
- **Second view** is called logical view and it describes static software models (classes, interfaces etc.) and dynamic relationship of different software models of the system. This view emphasizes inner functionality of the system.
- **Third view** is called process view and this view describes interaction and organization between parallel processes and threads. This view emphasizes performance, scalability and distribution of the system.
- **Fourth view** is called implementation view. This view divides system to physical parts (files etc.), which are gathering together to be represented in special way. This view emphasizes controlling of software product.
- **Fifth view** is called deployment view. This view describes hardware composition of the system; connections needed by hardware and software modules and processes locate in different hardware component. This view emphasizes distribution of the system, delivery of the system and assembly of the system.

Using software architecture gives quite good new concepts like classes, interfaces, components, modules, sub-systems, inheritance of classes, processes, messages, files, hardware components, communication models etc. Architecture model may concern either static structure of software like classes or dynamic structures like objects. It may also concern static relationships as well as dynamic behavioral models. Architecture can be described either as abstract model, which has not direct relation to software, or as concrete model. An example of applying the views in mobile music service will be presented in **Figure 3**. [4][6]

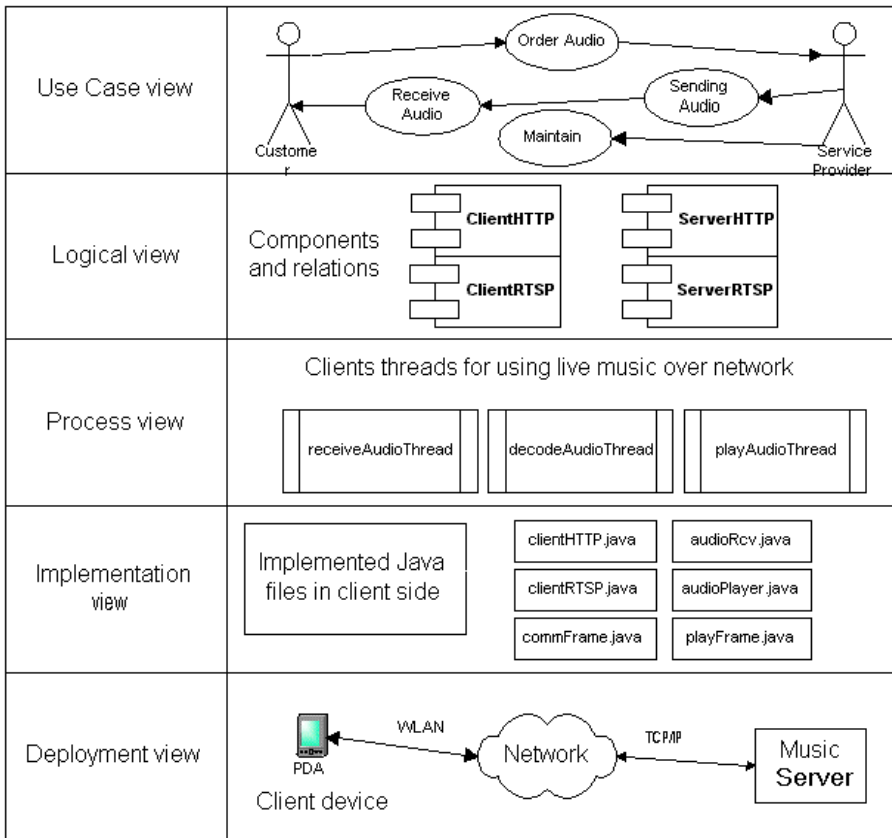


Fig. 3. An example of different views in mobile music service

4 Experiences of a Mobile Service Development

The experiences presented here are based on mobile card-game application development during MONICA¹ and MOOSE² projects. The projects have theoretical proportion where concepts in this area are studied and practical proportion in which the concepts were tested in real mobile application development. As a case example of mobile service a game service development was chosen. There were number of reasons for the decision; for example entertainment business is one of the most growing part of the mobile business and games have always being the best test bed of new technology.

¹ The target in MONICA project was to improve production facilities of value added services in mobile telecom area. More information: www.monica oulu.fi

² MOOSE-project is studying software engineering methodologies for embedded systems

4.1 Card Game

The card game chosen for the development experiment is called „Tuppi“. Tuppi is a card game from northern Finland and lumberjacks traditionally play it in their spare time. It is similar to Bridge game and can be referred as „Arctic Bridge“ or „Lumberjack’s Bridge“. Tuppi is played by 2 to 4 players. Players are in two teams. The aim of the Tuppi-game is to play rounds and collect tricks. The team that reaches 52 tricks is the winner and the other team is said to be in Tuppi. Being in Tuppi is thought to be very embarrassing to the team. There are different kinds of objectives inside Tuppi-game to define how tricks are calculated: these are called „Rami“, „Nolo“ and „Solo“. The game objective is chosen in the beginning of each round. More information about Tuppi can be found on the Internet: <http://www.geocities.com/TimesSquare/Arena/5911/>.

4.2 Development Baselines of the Mobile Card Game Architecture

There was couple of baselines that were taken into account in the projects. Understanding the design and development process of mobile value added service was the number one. Emphasizing on the reusability right from the beginning and developing by applying an abstract architecture design for the mobile service was the second main baseline. Applying these development baselines affected the definition of the software architecture of the application. When examining the architecture according to different architectural views of software it was quite easy to analyze what effects these development baselines chosen had into the resulted software architecture.

4.3 Use Case View of the Mobile Card Game

In use case view the system is looked from the user viewpoint. During the card game development a sample of case studies were used to describe the system from the user perspective. The case studies helped quite much in finding out additional requirements and new feature ideas. In use case development an approach defined by Dzida and Freitag [1] was applied.

According to this approach use cases have three perspectives: use scenario, indented system use and context scenario. Use scenario describes the use situation. Who is using the system and what is the motivation of the user. Indented system use tells us what kind of system we are dealing with and for what purpose for we are using it. Context scenario describes the backgrounds of the use and the context in which the use scenario is valid.

In the projects a metaphor of game house to represent Mobile Tuppi was generated. In game house players may meet and play different games. The game house architecture is described in the subsequent Figure 4 from the use case view. New player has a key to the game house where he can chat with other players in the lounge. When they decide they can go to one of the game rooms and start to play a game. Game rooms have tables where players have card decks and cards. The room, where the player is, defines the game that can be played. New games are added by adding new game rooms. This was done in Mobile Tuppi by defining rules for the

new game. Players can have cards on the hand and other cards on the table. This metaphor is well descriptive since the objects in the metaphor are the actual objects in the logical level of the architecture description.

4.4 Logical View of the Mobile Card Game

During the MONICA and MOOSE projects different distributed mobile client-server systems were studied and found that there is quite similar basic functionality in many of the systems. Therefore, there are similarities in the architectures. When considering the application from the logical viewpoint it can be found that the most of the components that are used are the same. Major differences in server and clients are in the components that manipulate and present data instead of entire architectural structure of the application system.

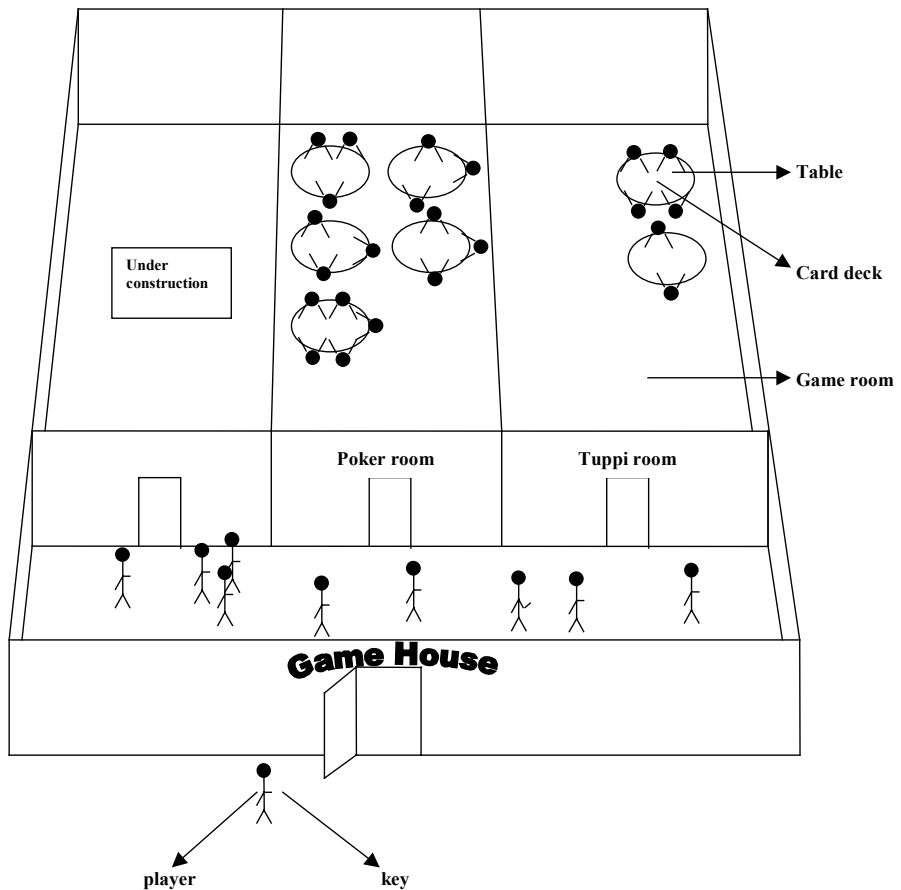


Fig. 4. Use case view for a Game house –architecture

When defining a reusable architecture solution it is essential to differentiate those components from other components. The use of layered architecture in server helped this differentiation. The higher level the layer is described, the closer onto that layer becomes the independent application. Layered software architecture gives a clear and simple structure with a possibility to develop and understand software as sets of independent layers. Previous solution makes a possibility to reuse most of the layers in other software development. For example communication classes are just focusing on transmitting data, they don't take a stand for the transmitted data itself. This approach emphasis data, data structure and data manipulation as the key points to consider when aiming towards abstract reusable architectures.

Mobile Tuppi-server architecture consists of five main objects: game server, serial gateway, router, encoding server and wireless access server. These objects can physically reside in same device. Task of the game server object is to administrate games, players and game sessions and to handle actual play of a game. When clients connect to the server each player will be identified with a session key. When creating new game players are assigned to it. Game server also keeps track of the client connections and automatically manages error situations such as network problems and lost connections.

In order to make communication between server and client more abstract, a special game language was developed. This language tells to the client what cards are on the deck, what are on the table, who are the players, whose turn is it, what game is it and so on. If something like XML language would be used instead of the game language, the architecture would be even more flexible. This would allow changing an application just by changing the components that manipulate and present data.

General architecture of the Mobile Tuppi client represented in Figure 5 consists of three parts. These are communications object, repository or shared memory and display object. The architecture is called repository architecture. Communication object handles different communication routines: opening and closing connections, transferring game information from client, and handling different communications errors. Shared memory is accessed both by communication object and graphical object. This object is used to store game information. Display object consists of graphical objects and routines that are visible to the user. The object handles user input and output. Repository architecture was used for clients because of limited resources were available on client platforms and this enabled to use more simple functionality than in the server. An abstract architecture definition helped here to port the application onto new client platforms quite easily. If the architecture used is well-abstracted, different components and their interfaces become quite clear. It is also easy to change implementation language if needed, but still to use the same architecture in different mobile platforms.

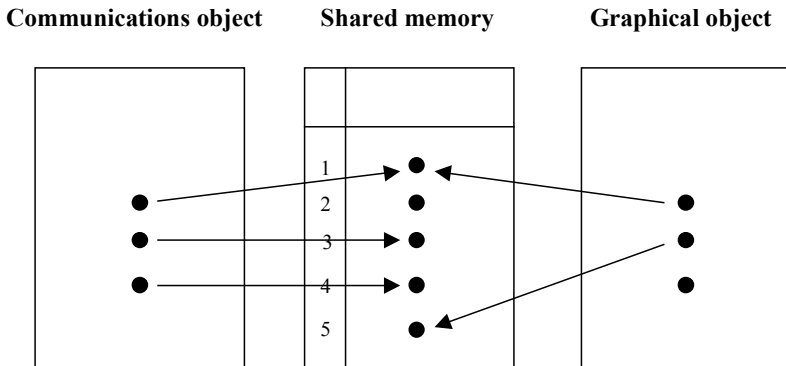


Fig. 5. General Tuppi-client architecture

4.5 Process View of the Mobile Card Game

Tuppi clients have two different threads. Communication classes are run in one and user interface and data manipulation classes in other. These threads communicate through blackboard or data repository. It is easier to distinguish data and data manipulation from the rest of the application by using repository architecture in the client application. This makes client application more abstract and reusable. For defining new applications you just change the parts of the client that handles UI and data. Other benefits are better performance since communication routines do not have to wait for the UI routines to finish and easier synchronization between communication classes and UI. This is quite important when working with devices of very limited resources like mobile phones.

4.6 Implementation View of the Card Game

Implementation view to architecture consists of physical parts (files etc.) of the software. Building mobile multi-platform software makes this view challenging, because software has different files for different platforms, and different distribution channels for them. Mobile Tuppi that supports a variety of platforms has different files for MIDP implementation, Java implementation etc., and number of ways to deliver them to different platforms. The MIDP applications can be delivered to the mobile terminal with SMS message and installed automatically.

At the time of the case study of mobile technologies didn't support dynamic downloadable libraries or components. Client applications were delivered in single packed file. This is yet relatively simple. When component technologies can be used in mobile platforms you can have dynamically downloadable components and libraries. You need to have a good plan how to organize them.

This view for architecture has the advantage in large and complex applications where number of delivered files can be thousands and you need to plan the structure of the files. In addition to that there is a need to plan, how to maintain the files, and how to deliver them to customer.

4.7 Deployment View of Mobile Tuppi

Mobile Tuppi is designed to be a multi-platform multi-player game. Connection is possible from game system with various platforms running under different operating systems. Game system can run many simultaneous games played by number of people. Mobile Tuppi supports PC-clients, different mobile phones, Palm devices and Java enabled digital TV. Supported languages are Java, MIDP and native Epic. Different platforms can also connect to game system using different communications protocols and connections. At this point game system supports serial line, infrared, TCP-IP, WAP and HTTP protocols. Supported connections are serial line, infrared, TCP-IP, GSM and GPRS. In the future Bluetooth is also thought to be added to the list of supported bearers.

Planning the software to be flexible and re-usable from the beginning of the development proved to be on the right track. It was possible to start from very simple implementations and add more complicated platforms as the technology developed. When using layered software architecture, it was possible to add new bearer in a quite simple way, just by adding new modules for one layer and leave the other layers untouched. Figure 6 shows the deployment view of Mobile Tuppi. [2][7][8]

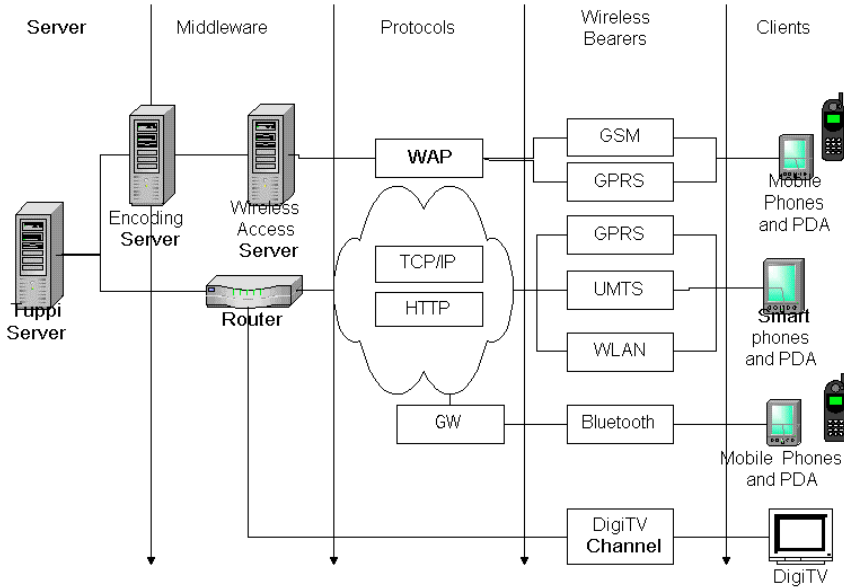


Fig. 6. Tuppi-game architecture

5 Tools Used during the Experiment

Tools that were used in mobile card game development consist of modeling tools (Rational Rose), programming tools (KAWA, Visual Studio), platform specific tools, finishing tools and from emulators (PALM, Symbian OS etc.) and devices (palm, mobile phones etc.). Platform specific tools and finishing tools are used for code pre-verifying, packaging and exporting to environment running software in portable clients. See Figure 7.

In the beginning a study of different development tools supporting development for mobile applications was performed. Different tools both for programming and modeling was explored. Mobile platforms set special requirements for application development on the behalf of memory handling, building graphical interfaces and communications. There are a variety of solutions for mobile application development. Many tools for development support only standard solutions and it is not possible to add new libraries for mobile development. Some tools however support this kind of possibility, but configuring them for mobile application development is a tricky task. Mobile solutions are developing fast and development tools must be kept up-to-date. There is a need for tools that support the use of standard description methods also for new mobile applications. In the ideal situation the whole software architecture can be modeled and programmed with the integrated CASE environment that supports automated code generation for different solutions, debugging and reverse engineering. This makes it possible to easily maintain architecture description for the entire mobile service.

Rational Rose was selected for modeling tool as it supports both C/C++ and Java programming languages. The process of development is a circular. First the base model for the game was done with Rational Rose and UML-modeling. This model was then engineered to code with Rose code generator. KAWA (KAWA IDE Integrated Development Environment) and Visual Studio was then used to further development of the code. Rose supports reverse engineering and that was used to keep the model up to date. If any changes were needed in the model, updated model was then engineered back to code. Code was ported to actual devices or emulators. Clients run a program in KVM (Kilobyte Virtual Machine), which is a Java Virtual Machine for Java for small and resource constraint devices. Different language libraries can be added to Rose so that it can do reverse engineering also for these new platforms. This made possible to make common architecture descriptions for both server and different mobile clients.

For Java development JDK version 1.3 was used together with J2ME (Java 2 Micro Edition), which is a modified Java platform for small devices. Early in the development KJAVA was used. It is a package of Java for PalmOS devices released at the 2000 in JavaOne show. It was renamed by Sun and is part of the J2ME packages. Figure 8 shows picture about Java architectures. Configurations define API's and features of the VM for the devices with certain amount of resources. For example amount of memory can limit which configuration can be used. Profiles define API's for certain domains such as MIDP profile for mobile phones and two-way pagers.[2]

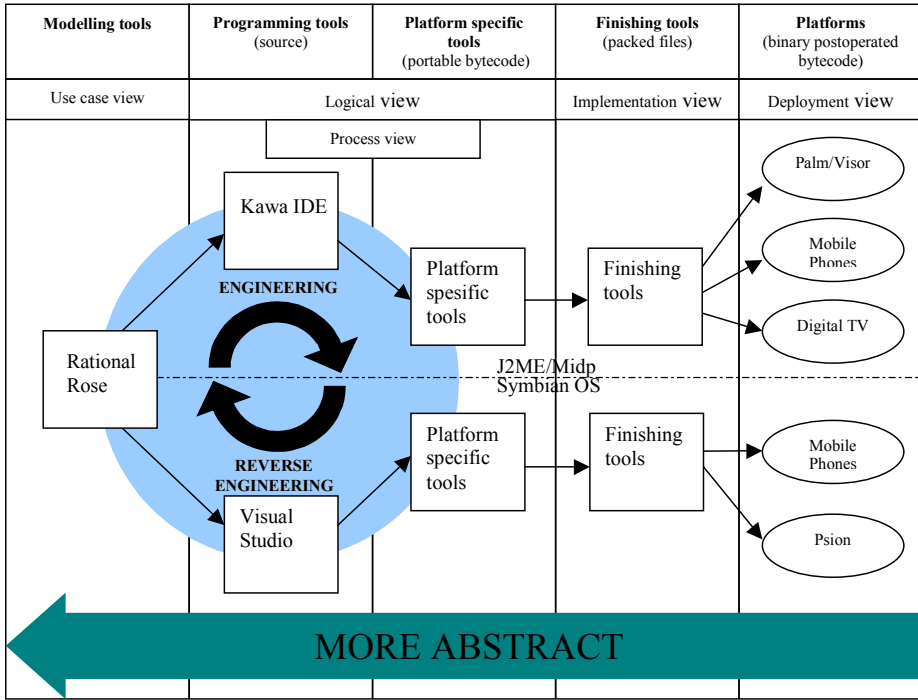


Fig. 7. Mobile application development tool chain in Monica project

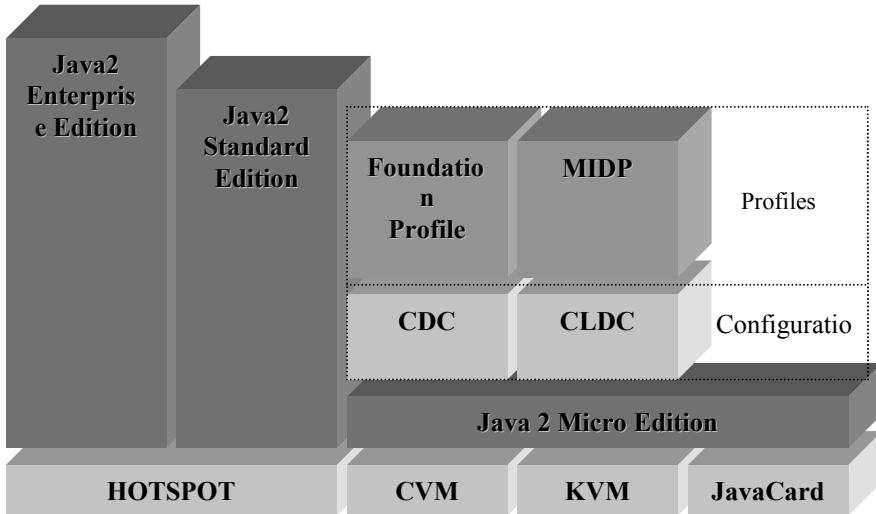


Fig. 8. Java architectures [2]

Symbian OS is customised C/C++ based development environment from Symbian that is used for example in Nokia phones. This produces native code applications that run faster than interpreted code in these devices.

6 Conclusion

It is evident that distributed technologies have a major role in future of mobile applications development. Mobile platforms have opened from restricted systems to allow use of third party software. Downloadable libraries and entities force developers to rethink existing architectures. Different architectural solutions are grouping into product line architectures that allow simultaneous programming of applications in several product lines for the platform in question. Trade of software and architecture solutions between different software companies will evolve since this kind of technology fits well in mobile software development business.

It is essential to success of the project to keep different stakeholders involved in software development process. Kruchten's model can be used to divide architectural information in different views for the same architecture. These views can be used as a tool for different stakeholders to communicate from the development. It also makes it easier to analyze the architecture from different viewpoints in order to create comprehensive solutions. This model can also be used to divide development tools with the same category see Figure 7. Traditional development methods works well in the development where goals of the software are clear from beginning of the project. This is not often the case in rapidly developing mobile business. This is why more flexible methods like prototyping are often used. Kruchten's model can be used to analyze traditional methods as well as prototyping but it does not suite well to agile methods that have totally different approach to development. Agile approaches are very interesting in mobile application development viewpoint. Their approach for development is emphasizing focusing on development and user involvement. They start with minimal documentation, try to produce only vital information and software features and reduce unnecessary documentation work (less is better). This kind of approach makes it easier to adapt to changing world and keep documentation and the system in synchronization.

Mobile application development belongs to the complicated, but quite regular software development area that includes many different solution possibilities in the development. The experiences from the mobile card-game development lead to the development process covering many distribution channels (different client platforms). Architecture and the readiness to react to changes that were caused by technology requirement changes became key points in successful development. Changes that came during the application development required clear architecture.

Identifying, standardizing and naming general architecture models helps to reuse same models and code in several applications. A normal life example about cars and roads instead of software business helps to understand how important is the re-use. New road is not build (infrastructure for content delivering) for every car (application) that is made. Same road is used that is made safe, well-defined and documented. This allows carmakers to focus onto their key business that is car making and users to safely use them. When new cars (platforms) were built the same road that before was used or there could be a new fast line (new bearers) to speed up

the driving. New roads are also built to go new places (new type of applications) but these are added to existing infrastructure. In the sense of software production this means that well-tested solutions are re-used for content delivery of different applications. This increases productivity, QoS (Quality of System) and keeps users happy.

Mobile Tupper architecture, in which data manipulation and UI are well isolated from the rest of the application functionality, give new ideas on how multi-platform applications may work. Different platforms have different capabilities. There can be platforms with limited memory and screens and multimedia platforms that have 3D UI with possibility to use speech and sound. During the card game development an idea came out to change the ways that the data is represented in different platforms by informing the application about the capabilities of the platform. This would mean content aware plug and play-architecture for mobile devices. With downloadable components new UI could be loaded to the phone when it is connecting to the system first time. There could even be different producers using same architecture that could do this UI component. Well-abstracted and documented game architecture could also mean that someone else could do new games or game creation could be automated (see also J. Orwant paper of automated programming for game generation [3]).

References

1. Dzida, W., Freitag, R.: Making Use of Scenarios for Validating Analysis and Design. *IEEE Transactions on Software Engineering*, Vol. 24, No. 12, December (1998)
2. Nokia: MITA, Mobile Internet Technical Architecture, IT Press, Finland (2001)
3. Orwant, J.: Automated programming for game generation. <http://www.research.ibm.com/journal/sj/393/part2/orwant.txt>, IBM (2000)
4. Koskimies, K.: *Oliokirja*. 2nd edn. Satku – Kauppakaari, Gummerus Kirjapaino Oy, Jyväskylä (2000)
5. Pressman, R.S.: *Software engineering – a practitioner's approach*. 3rd edn, McGraw-Hill Book Company, Berkshire, England (1994)
6. Kruchten, P.: The 4+1 view model of architecture. *IEEE Software*, volume 12, No 6 (1995) 42-50
7. Sun Microsystems: J2ME[tm] Mobile Information Device Profile (MIDP) <http://wireless.java.sun.com/midp/> (2002)
8. Lewins, J., Loftus, W.: *Java software solutions – Foundation of program design*, Addison-Wesley (1998)
9. Carnegie Mellon Software Engineering Institute: *Software architecture documentation in practice – Documentating architectural layers*. <http://www.sei.cmu.edu/publications/documents/00.reports/00sr004.html> (2002)
10. Carnegie Mellon Software Engineering Institute: *A Framework for software product line practice – version 3.0*. <http://www.sei.cmu.edu/plp/framework.html> (2002)
11. Seetharaman, K., *The Corba Connection*, *Communications of the Acm*, volume 41, No.10 (1998)
12. Sun Microsystems: *Java Remote Method Invocation (RMI)*, <http://java.sun.com/products/jdk/rmi/> (2002)