Overview
This techpaper takes a look at the Ameyo Platform from a developer’s perspective. It analyzes various application development aspects that ensure faster development of secure, reliable and scalable applications. For more details on Ameyo Architecture, please refer to DACXComponentService - an Architectural Overview.

Ameyo Development Platform focuses on:
• Producing better managed and tested code for increased reliability.
• Shortening application development lifecycle.
• Facilitating third part application integration and interoperability.
• Flexibility and customizability.

Architectural Considerations

Service Oriented Architecture
DACX Components leverage the SOA\(^1\) paradigm to build composite applications that draw upon functionality from multiple sources, within and beyond the enterprise. Service Orientation of Ameyo Platform enables:

• Focused Multi-layered Development: The application design has been divided across multiple layers viz, core services layer, framework services layer, and application layer. The segmentation enables each component to be developed and tested independently by a team of developers specializing in the particular domain. For example, a developer working in the fundamentals of distributed architecture like messaging, redundancy, and failover, and need not be aware of how to develop a user interface.

• Better Testing/Fewer Defects: All components have published interfaces that can be unit-tested easily by developers. These test suites can be run to validate the service independently from any application that uses the service. More and better testing usually means fewer defects and a higher overall level of quality.

• Application Composition: Business applications can be composed using pre-built components providing specific services. This results in better application customizability and configuration. Further, it provides the customer an option to quickly customize the application with a set of components developed specifically to cater to his requirements without affecting the standard components.

• Better Maintainability: Software archaeology\(^2\) is the task of locating and correcting defects in code. By focusing on the application layer as the location for business logic, maintainability increases because developers can more easily locate and correct defects.

• More Reuse: Code reuse has been the most talked about form of reuse over the last four decades of software development. Unfortunately, it is hard to achieve due to language and platform incompatibilities. Component or service reuse is much easier to achieve. Run-time service reuse is as easy as finding a service in the directory, and binding to it. The developer does not have to worry about compiler versions, platforms, and other incompatibilities that make code reuse difficult.

• Easier Application Integration: Clearly defined contracts of Components help in easier integration of DACX Components with third party tools like CRM Solutions, Communication Media, Enterprise Workflow Solution, etc.

Plug-in based code development
Ameyo Development Platform inherits the benefits of industry-accepted dynamic component framework, OSGI\(^\text{TM} \)^\(^3\) developed by OSGi Alliance. OSGi technology is the dynamic module system for Java\(^\text{TM} \). It provides the standardized primitives that allow applications to be constructed from small, reusable and collaborative pieces. These components can be composed into an application and deployed. DACX Components effectively use the OSGi bundles to organize and maintain the software code base. This provides:
• Dynamic management of bundles at runtime including installation, activation, upgrade and uninstallation
• Bundle versioning and dependency management.
• Concurrent deployment of multiple versions of bundles.

Model Driven Architecture
MDA is a software design approach intended to support model-driven engineering of software system. It provides a set of guidelines for structuring specifications expressed as models. Ameyo Platform supports MDA’ based application development. Its data and logic separation approach and SOA based component...
contracts make it suitable for MDA development. Model documents from various applications like Rational Rose®, Eclipse EMF, UML can be imported within the Ameyo Development Platform and used to generate DACX Components.

Ameyo IDE provides tools to support various MDA tools including:

- Creation and editing of models (in EMF)
- Analysis and validation of models
- Composition of models
- Simulation of models
- Meta-model management for relationship management

Model Driven Declarative User Interface

Ameyo Declarative User Interface provides a reliable and faster way of developing user interfaces. The approach creates a User Interface Model (UIM) by combining the Application Model (AM) with Generic User Interface Model (GUIM). GUIM consists of

- User interface widgets like text, combo, tables, trees, tool tips etc
- Action defining model
- A rendering engine to render the same

Ameyo User Screen Designer, a GMF based editor, is used to design the user screen graphically. Data-binding of the widgets is provided declaratively using AM mappings. AM is independent of the platform and is application specific. These models are bound together by Ameyo Declarative User Interface, which provides:

- Dataflow management based on models (using reflection)
- Data binding between UI widgets and application
- Actions and triggers
- Communication with Ameyo application servers

Instances of composed UIM then represent the User Screens in model. The rendering, data fetching and actions are then managed by framework.

Declarative User Interface Framework helps in

- **Faster Development**: The graphical designer and declarative way of populating data from the model makes User Screens design and development a child’s play.
- **Reliable Development**: Since the framework handles data binding and communication flow centrally, it ensures that the execution flows are properly executed and errors are handled properly.
- **Faster Changes**: Since the interface design now consists of changing data-bindings and UI model does not require much UI expertise, it can be executed by application consultant. It thus saves a lot of time and expertise which would have otherwise been invested in changing/ managing the UI code. Thereby enabling drastic reduction in customer turn around time.

  - **Separation of Roles**: Enables clear separation of roles among User Interface designer who can now focus on incorporating new and innovative UI designs and concepts like skinning, widgets etc and Application User Interface designer who uses these components to provide the business application.

Simulation Based Development

Ameyo Development Platform provides powerful scripting languages to simulate the various functional modules of the system. This enables simulation based development of a group of integrated components which is impossible to do with a unit-tests simulator. For instance:

- **SSPL** can simulate an application server, thereby supporting independent development of the entire client including the user interface and server communications.
- **CSPL** can simulate client behavior including telephone pickup, agent actions and customers. This enables complete independence from the actual client during server development.

Script based Simulation empowers Ameyo Platform with

- **Faster and Reliable Development**: Simulation environment for client helps to remove the dependencies with the server team and vice versa. Also the scripted nature of the simulation helps in creating scenarios which are otherwise very difficult to recreate like synchronization issues.

- **Rapid Prototyping**: With the server simulator, user interfaces can be developed using the user screen designer with the simulated application model. This enables rapid prototyping and customer feedback on new features. This potentially saves a lot of actual development time.

Development Considerations

Code Generation

In its quest for faster and reliable application development, Ameyo Platform heavily uses code generation. In fact most of the code that can be generated is auto generated including the application model code, communication code, stubs, services stubs, HTTP command wrappers, client proxies etc.

Code generation is a proven technology to write quality code and Ameyo Platform uses it effectively to:
• Produce quality code and reduce chances of human error
• Be standards compliant wherever applicable
• Be flexible to accommodate changes in the application requirements
• Provide flexibility in terms of technology and deployment options

Automated Testing
One important aspect of reliable code is testing; more rigorous the testing, more reliable the code would be. Ameyo provides tools and environment for automated testing of components including:

- Unit Testing of components
- Integration Testing with simulation
- Scenario and Stress Testing using scripts

Ameyo IDE
Ameyo IDE included in Ameyo Development Platform is an Eclipse® based Integrated Development Environment. Eclipse is an open source community, whose project are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle.

Ameyo IDE enhances the Eclipse IDE with support for Ameyo Component development, code generation and MDA® tools. It enhances the power of Eclipse to greatly increase the developers' efficiency and reduce development time.

Object Relation Mapping
Object-relational mapping is used to map Object-Oriented Programming (OOP) object to relational databases managed by Oracle, DB2, Sybase and other relational database managers (RDBMS). Ameyo supports ORM using Hibernate. It helps to
- Provide a transparent persistence using a RDBMS (oracle/MySQL/DB2 etc)
- Reduce programming code by abstracting driver code
- Improve performance through caching over using an embedded SQL or call-level interface such as JDBC or ODBC when using an RDBMS

Debugability and traceability
Ameyo Platform provides extensive features for debugging and tracing by turning on Logging, which encompasses exhaustive data logging including timestamps, node-id, flow contexts and application contexts. This can be coupled with intelligent log processor to extract relevant information. For example
- Flow during an event processing that spans across multiple machines and threads.
- Business context based correlation. For instance, tracing a user would involve tracing the user activity, his telephony activities and other activities which have only business rule based correlation
- Stack-traces of exceptions and errors

Thus it facilitates easier debugging and tracing both at programming and semantic (business logic) levels.
References

1. DACXComponentService-An Architectural Overview
11. http://www.hibernate.org