A New Digital Convergence Technique in BSS – A Case Study of Integrated Telecom Order System

Chiao-Yi Wang, Ginn-Feng Liu, Chun-Chen Chang, and Chih-Ying Liu
Telecommunication Laboratories Chunghwa Telecom Co., Ltd.
Taipei, Taiwan, R.O.C.
joykate@cht.com.tw

***Abstract*—While the liberalization and globalization of telecommunication market has already become a worldwide trend, a cut-throat competition is inevitable. To provide reliable and high quality telecommunication services to meet all manner of customer demands promptly is not only very important indeed but also must rely on effective heterogeneous resource integration foundation. However, due to the resource peculiarity of each telecommunication business, such as fixed network, mobile, digital service, and the latest cloud service, integrating those heterogeneous resources will be a tough mission and hard to be accomplished. Though building a compatible application system to solve such plight will be highly difficult, eXtreme Order (XORDER), which is an integrated telecom order system, takes the lead in adopting NGOSS solution framework and policy-oriented framework to provide a more resilient and extendable solution to respond to all sorts of customer demands immediately as well as some other oncoming challenges.**

***Index Terms*—Automatic pricing, Automatic user interface generation, Digital Convergence, NGOSS, Policy-based, Product management and fulfillment, Rule-based, Shared Information/Data Model, SID, SOA, Telecommunication.**

I. Introduction

As telecommunications markets are liberalized around the world, customer requirements are not only increasing but also drastically changing. To effectively satisfy what customer expected and even lead customers' demands further, conventional telecom services are definitely insufficient to meet. For the sake of staying competitive, operators are eagerly looking for approaches for efficient service creation which would allow for reduction of time-to-market as well as reduction of development and operation costs [7]. The integration of heterogeneous resources, services, products, discount information, and ordering and pricing mechanisms will be inevitable as well as extremely urgent for modern telecommunication companies.

Though the peculiarity of each telecommunication business is quite dissimilar, many additional special restrictions and complex business logics will be derived from their own responsible unit to result in a difficult integration of those heterogeneous resources. Thus common design patterns in application development are inappropriate and even need to be avoided because they often mix the application implementation logic with business logic, user interface logic, and even data access logic [5], for example, "hard-coded". In consideration of foregoing problems, a novel and intelligent telecom order system will be necessary and it must be able to provide more reliable, prompt and agile mechanisms so as to conquer each oncoming challenge, such as integrating discount packages, pricing flexibly, automatic user interface generation, instant data validation, and even producing correction records dynamically.

This paper introduces our approach, discusses its capabilities and presents our experience. The focus of our research group was not limited to conceptual development only. As a proof, there will be some examples explained how to apply different policies to generate user interface and price dynamically.

II. Related Works

Customers always change their requirements, therefore programmers will be stuck in endless system development cycles. When it comes to requirement variations, changes for system programs especially user interface are usually unavoidable. Except for the importance of the separation between business logic and development logic had been mentioned in a few literatures [1][2][3][5], several approaches for solving such problem had also been proposed.

To end iterative system development process and provide an overall solution, a policy-oriented system development framework [3] had been proposed but still incapable for building an integrated telecom order system. To begin with, we found out that managers or users are incapable of editing whole policies by themself, the better way is to provide primitive policies and then modify further. Thus the PolicyGenerator for transforming static specifications into primitive policies will be necessary. Next, in view of resource integration, figuring out each potential information domain will be contributive to define policies. Thus, we adding the PolicyAction on behalf of different information domain, provides available instructions for editing policies, and even capable of dealing with cross-domain's logics. Furthermore, the interaction between UI fields are sometimes quite complicated (e.g. cause and effect logics), there should be an independent mechanism for handling, so the CorrelationEngine proposed in this paper would be more useful. Finally, producing correct correction records is important but also laborious. Therefore, we add an efficient data transaction monitor to systematically control every data change event and even generate correction records automatically.

III. Policy Framework

In order to stay competitive and adapt to the market variations, we proposed a refined policy framework that consists of the following components (Fig. 1) for timely and appropriately controlling operating strategies:

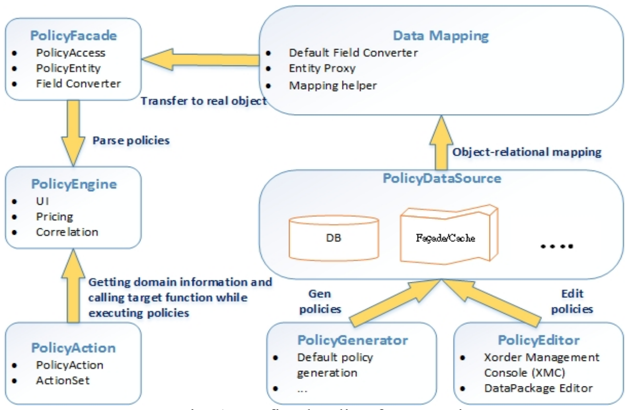

Fig. 1. Refined policy framework.

A. PolicyEngine

For different purposes, we built kinds of engines as below:

1) UIEngine

Dynamically generating user interface.

2) PricingEngine

Flexibly pricing, calculating and validating fields.

3) CorrelationEngine

Dynamically validating fields and triggering updating.

B. PolicyFacade

An immediate access layer that in charge of the following jobs:

1) Policy access

PolicyAccess is the only subcomponent of PolicyFacade which could really accessing and fetching policies.

2) Policy entity definition

There are several PolicyEntities has been defined for describing customer requirements or business logics in detail. In general, a requirement or logic may need more than two kinds of PolicyEntities to describe.

C. PolicyAction

1) Define what may be done by each policy

Each kind of PolicyAction not only stands for a business domain or dimension but also the real executable unit because each of them possesses its own functions to deal with its domain's exclusive logics and makes policies more expressive.

2) Collect sufficient but exact information for executing

ActionSet is a group of several PolicyActions and is general called 'api' for short. Grouping PolicyActions is for the sake of preparing sufficient but exact domain information before triggering policy engine. Each PolicyAction could be included in different ActionSets at the same time.

D. PolicyDataSource

Providing a policy repository which must be a free format media with open design adopted. As long as fit the access standard of PolicyFacade, any media can be used to as the entity/policy data source.

E. DataMapping

Adopting an object-relational mapping alike technique to enable PolicyEngine to dynamically execute what the policy described just like operating an object instance directly.

F. PolicyGenerator

Automatically transforms static requirement specifications (e.g. SID) into effective and more expressive policies.

G. PolicyEditor

We build XORDER Management Console (XMC) for managers to further edit or modify policies if he/she thinks the existing policies are insufficient or incorrect, especially requirements changing.

IV. XORDER Architecture

Xtreme Order (XORDER) is an integrated telecom order system that follows TeleManagement Forum's New Generation Operations Systems and Software (NGOSS) solution framework design, the four core elements are as below:

A. Telecom Applications Map (TAM)

TAM is used to design and plan all OSS/BSS systems related to any process of the whole business lifecycle, such as pre-sale, selling, service configuration and activation, and maintenance.

B. Enhanced Telecom Operations Map (eTOM)

The Business Process Management (BPM) was adopted to establish a set of Standard Operating Procedure (SOP) to plan detailed system functions and even overall procedures.

C. Shared Information Data Model (SID)

We adopted SID as the standard specification of describing requirements and the criterion of data exchanging among different systems to ensure all operating procedures will be fulfilled effectively and smoothly.

D. Technology Neutral Architecture (TNA)

XORDER adopts Service-Oriented Architecture (SOA) and Microsoft service platform to build a service management mechanism to effectively control the overall operating procedures. The architecture of XORDER is been arranged as the following layers (Fig. 2):

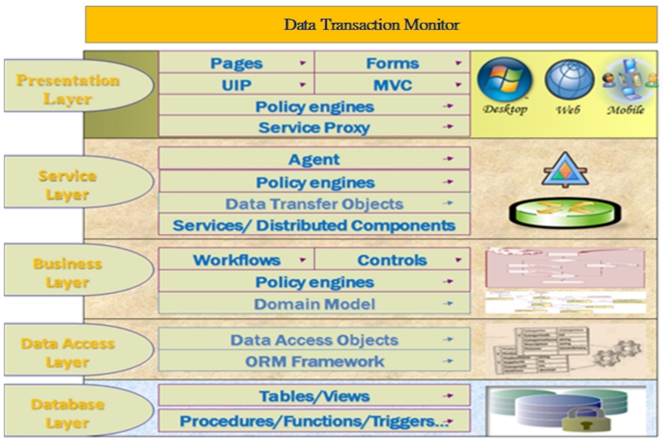

Fig. 2. XORDER architecture.

V. Policy Setting and Execution

Flexibly pricing after taking orders is a tricky issue for telecommunication order system because it is hard to decide how much data is necessary and needs to be collected in advance. Based on our refined framework, all we needs to do is to figure out what kinds of business information may be needed, thus we can define a specific ActionSet for pricing. To illustrate further, the following are some crucial requirements of CHT SaaSCRM product and their related policies.

1) Requirement 1

If the customer has already used 3G mobile service in superior contract types (e.g. mPro 450, mPro 750, mPro950) then he would be allowed to buy CHT SaaSCRM product in the most preferential contract (discount code: Vis003).

2) Requirement 2

Originally, the monthly rental of CHT SaaSCRM is NTD$1,000. When customer chooses the first contract type (discount code: Vis001), the monthly rental is 5% off. When customer chooses the second contract type (discount code: Vis002), the monthly rental is 20% off. When customer chooses the third contract type (discount code: Vis003), the monthly rental is 50% off.

To achieve dynamic UI generation and flexible pricing, we can easily transform above requirements into some manageable policies in Table3 and Table4:

Table. 3. Policies for agreement presentation_ SaaSCRM

FieldName	**Candidates**	**Cond**
DiscountCode	Vis001, Vis002	api.MBMSBusiness. mProType(api.Customer.Id)<=1
DiscountCode	Vis001, Vis002, Vis003	api.MBMSBusiness. mProType(api.Customer.Id)>2

Table. 4. Policies for pricing_ SaaSCRM

FeeName	**Fixed Price**	**Selling Price**	**Cond**
MonthlyRental	1000	950	api.Agreement.IsEqualChtCode(“DiscountCode”, “Vis001”)
MonthlyRental	1000	800	api.Agreement.IsEqualChtCode(“DiscountCode”, “Vis002”)
MonthlyRental	1000	500	api.Agreement.IsEqualChtCode(“DiscountCode”, “Vis003”)

Table 3 list two agreement-related UI presentation policies. In which,”MBMSBusiness” is a kind of PolicyAction which used to cope with all the heterogeneous business logics of mBMS. “mProType” is a specific function provided by MBMSBusiness and mainly used to check whether the customer has used any CHT’s 3G mobile service.

During the process of editing agreement data, UIEngine will first retrieve all agreement-related UI presentation policies and check each one’s condition. When it comes to “api.MBMSBusiness.mProType”, UIengine will immediately focus on current ActionSet's MBMSBusiness action then execute its “mProType” function and return an instant result: 0, the customer has not used 3G mobile service yet; 1, the customer has used mPro150 service; 2, the customer has used mPro450 service; 3, the customer has used mPro750 service; 4, the customer has used mPro950 service. After get a certain answer, UIEngine will further apply valid policies’ setting to UI fields. For example, if the first policy in Table 3 is valid, user could only see “Vis001” and “Vis002” two candidates when check DiscounCode field's dropdown list. Otherwise, user could find out additional candidate “Vis003” only if the second policy's condition comes true.

Table 4 list some pricing-related policies. In which, “Agreement” is a kind of PolicyAction and used to cope with all agreement-related logics. “IsEqualChtCode” is a function of Agreement and often used to check whether the target property’s content value is equal to the target code.

When get into the process of pricing, PricingEngine will first retrieve all SaaSCRM-related pricing policies. When it comes to “api.Agreeement.IsEqualChtCode(property name, target value)”, PricingEngine will immediately focus on current ActionSet's Agreement then execute its “IsEqualChtCode “ function and return an instant result: true, the property is a ChtCode class object and its code or name is equal to the target value; false, the property is a ChtCode class object but both its code and name are not equal to the target value. Therefore, when customer chooses “Vis001” discount code, the first policy in Table 4 will be valid and the monthly rental will be set to $950. When customer chooses “Vis002” discount code, the second policy in Table 4 will be valid and the monthly rental will be set to $800. When customer chooses “Vis003” discount code, the third policy in Table 4 will be valid and the monthly rental will be set to $500. Moreover, we can even make sure that only customers that has already used superior 3G mobile service were allowed to buy SaaSCRM with so favorable price.

VI. Conclusion

The paper presented an integrated telecom order system that using the policy techniques to realize the goal of NGOSS solution framework. In order to provide a more reliable and agile mechanism, the technique is been enhanced by the PolicyGenerator, which can automatically convert static requirement specifications into primitive policies and enable the general managers or users could easily edit policies and comprehend. Next, the PolicyAction clarifying potential information domains and increasing the flexibility of policy expressiveness and extensibility. When a new information domain emerges, creating a new PolicyAction stands for it, then could simply bring about its related business logics. Furthermore, the CorrelationEngine improves the ability of dealing with complicated interaction issues or restrictions, for example cause and effect logics. Finally, the data transaction monitor achieve the goal of precisely observing each data change while user modifying the order and then automatically generating correction records after the whole transaction committed. Thus, the development logics and business logics could be clearly demarcated, the system could simply integrate information from any product or business, and the goals of flexibly pricing and user interface generating, integrated marketing, and even digital convergence could be achieved assuredly.