International Journal of Smart Home
Vol. 1, No. 2, July, 2007

A Reconfigurable Software Distribution Framework for Smart Living Environments

Ing-Yi Chen
Department of Computer Science and Information Engineering,
National Taipei University of Technology, Taipei, Taiwan, 106, ROC
ichen@ntut.edu.tw

Chao-Chi Huang
Department of Computer Science and Information Engineering,
National Taipei University of Technology, Taipei, Taiwan, 106, ROC
brouse@ilab.csie.ntut.edu.tw

Abstract

This paper seeks to define a process that allows for the reconfiguration of management processes, in a manner that is easier than those afforded by existing approaches. This project proposes to achieve this through the use of a Software Distribution Management System (SDMS) that employs a service-oriented mechanism. This constitutes an improvement over more traditional distribution systems because SDMS allows for the reconfiguration of management distribution processes. In addition, this study applies a Service-Oriented Software Engineering (SOSE) development methodology for designing and building of a Service Oriented Architecture (SOA) based on SDMS. While based on a conventional methodology this gives consideration to added dimensions such as service choreography, and service bus. The result is a approach that allows software engineers to readily reconfigure and recombine services from a range of different systems, applications, architectures and ages. This paper provides the service identification results and monitoring methodology of the SDMS produced by this project.

1. Introduction

In recent years, technological innovations have necessitated a shift in software distribution. Software Distribution [1] is a generic term (also called Software Delivery [2] etc.) for systems that can be used to configure, update, and manage applications. Primarily, this shift has entailed a movement in focus from central servers to remote PC's and ubiquitous environments where applications are executed.

Since the 1990s, software distribution research work has focused on improving service quality [3]. Scholars have been making advances in the field of software distribution for decades. The resulting systems can be categorized into three classes: central servers, remote PC's, and pervasive devices in smart living environments. These categories also constitute an evolutionary pathway.

Traditional software distribution systems research, such as that of Geng [4] has introduced the distribution management framework used in managing servers, particularly server provisioning. Over time, this has resulted in different application versions installed on various PCs operating simultaneously throughout various companies and organizations. This has forced IT administrators to track and support multiple versions of dozens of applications.

Briskin, G. [5] work caused technology to evolve beyond traditional servers, and towards software distribution for PC's. While PC distribution constituted an improvement over server based technology, the growing use of mobile devices has made its fixed locations increasingly limiting. This is causing a revolution in the requirements of services distribution.

Two important infrastructures for mobile software distribution management were put forward by Andersson [6] and Topalis, E. [7]. Since 2004 Topalis's home gateway application has driven a revolution in software distribution. Home gateways can be updated with a range of new services. Subscribers are able to update applications on-demand, rather than waiting for a service provider to do it for them.

Even when solutions for home application distribution are in place, some problems remain. One problem has been the issue of adapting distribution applications to various needs. Such problems persist because a gap exists between the software distribution engine and the domain requirements.

In the past, software distribution systems were typically designed to meet specific goals rather than the strategic objectives of an application provider as a whole. Thus, their design architectures have made it difficult for them to communicate with each other and to make changes in response to new demands.

As these systems become more complex, two major needs have emerged. First, to improve service, there is a need to develop an architecture that can integrate the basic functions of software distribution with specific domain needs.

Second, there is a need for a value-added domain application that has the ability to integrate systems across platform boundaries. Most home application providers today use a range of different systems, applications, and architectures, all of different ages. Since software distribution engines and domain application are heterogeneous platforms, traditional software distribution approaches do not provide a framework that is adequately equipped to integrate them. This problem is compounded by the fact that integrating products from multiple vendors, across different platforms is almost always challenging.

The implementation of service-oriented architecture (SOA) using web service technologies is currently the approach of choice in systems integration. The defining feature of SOA is that it describes domain applications as a collection of processes that each span functional boundaries [8-10]. There has been extensive discussion on web services and SOA technology, in numerous papers. Hence, this paper proposes a software distribution management system for smart living environments, based on SOA, without engaging in an in-depth discussion of the framework itself.

It is perhaps sufficient to note that service-oriented architecture represents a critical new application development style in the software area. As noted, traditional software engineering methodologies are inadequate to permit the engineering and distribution of these critical new applications.

Service Oriented Software Engineering (SOSE) provides a methodology capable of overcoming many of the challenges of integration. There is some existent research on SOSE, including that of Harri Karhunen [11]. This project, then, seeks to propose an adaptable software distribution management server (SDMS) for home environments using SOA technology and Software Engineering methodology (Figure 1). In addition, this paper serves to provide a case study of the use of this framework in smart-home scenarios.

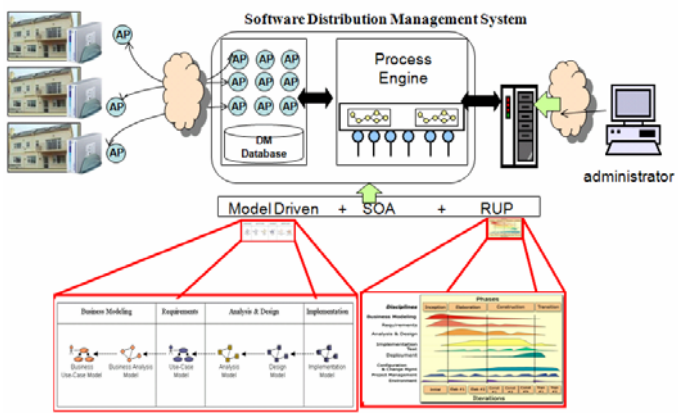

Figure 1. A Reconfigurable Software Distribution Framework

2. Related Work

Several existent modeling methodologies such as Object-Oriented Analysis and Design (OOAD) [12-13] were introduced in the early 1990's. These provided a disciplined approach for software analysis and development. They became common practice and eventually constituted a starting point for implementing distributed applications. While the OOAD methodology is a required precursor, it is not, in itself sufficient to produce SOA technology.

In the field of software development, the use of a UML model [14] has emerged as a premier methodology. However, RUP is built upon the principles of OOAD, and is limited on this basis, because it does not easily lend itself to being utilized in SOA design.

While these traditional methods reinforce general software architecture principles such as information hiding, encapsulation, and modularization, SOA introduces additional themes such as service choreography [15] and service bus [16] that require additional consideration. Hence, a key recommendation for development of a SOA is to adopt a solution development methodology that designs and builds SOA components.

3. Using SOA Development to Bridge the Smart Living Domain Gap

The proposed solution architecture uses SOA technology to connect the current distribution management server with home application needs. Conceptually, there are four major levels of abstraction within SOA:

1. Existing programs: are capable of working with existing systems.
2. Services: are logical groupings of home application management functions. They have well-defined interfaces that let service consumers know how to interact with them
3. Function Processes: are sets of activities performed with specific business goals in mind. Business processes typically encompass multiple service invocations.

4. Domain Applications: are function processes that are further combined into domain applications that fit specific home application requirements

Figure 2 represents a theoretical home application provider that needs to implement a home application in order to support customers' services. Since the provider already has a software distribution server it would prefer to reuse the existing functionality, rather than writing new home applications to replace the original system. In this project, the pre-existing software distribution server is called the software distribution engine.

Services are the building blocks of SOA, providing interfaces for which systems can be built for distribution. Services can be invoked independently by consumers to process simple functions. They can also be chained together to perform more complex tasks and, therefore, to devise new function processes quickly.

Furthermore, a home application distribution service can be defined as any discrete function that can be offered to an external consumer. Since each level of the framework is able to work independently, all levels of the architecture are potential service packages that can be acquired, combined and recombined as individual business needs arise.

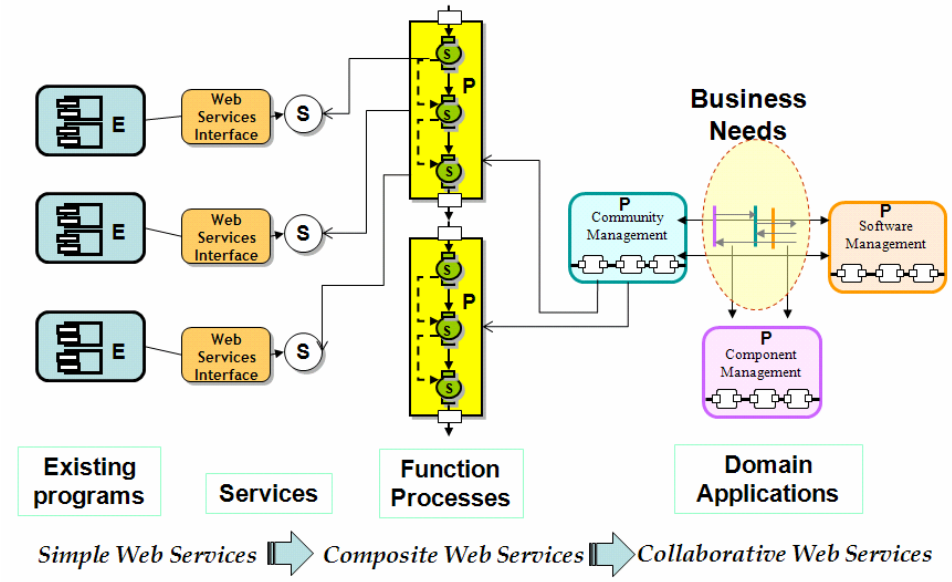

Figure 2. Tiers of SOA system

Hence, by adopting a SOA approach and implementing it using supporting technologies, software distribution service providers should be able to build flexible systems that allow for the speedy and efficient change of function processes. This is because the approach has the benefit of the extensive re-use of components.

4. The proposed System Architecture

Figure 2 shows the architecture of the SOA-based software distribution management system. This architecture consists of four parts: a distribution agent for communicating with the software distribution engine, the software distribution engine, a process module, and an operation management portal

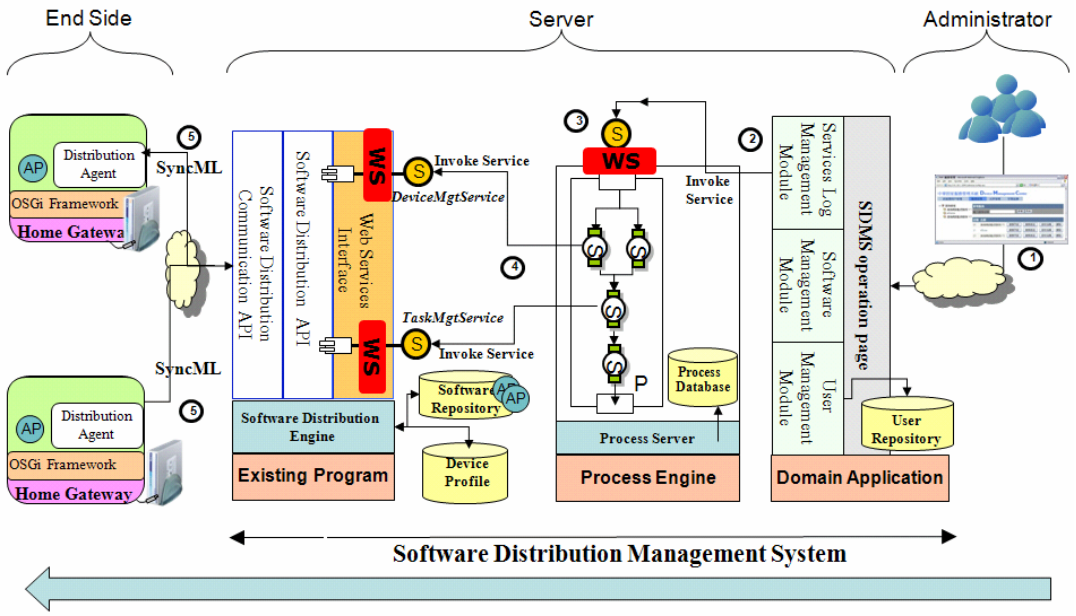

Figure 3. Architecture of the Software Distribution Management System

This project used an Intelligent Home scenario as the application domain. In the following representation (Figure 3), the existing software distribution engine is responsible for providing the ability to deliver applications to home gateways.

The distribution agent is an essential component of the home gateway that communicates with the software distribution engine. As soon as the distribution agent is configured, it can be enrolled as a subscriber on the software distribution engine. The agent can then automatically update itself periodically.

Due to the function gap between the existing software distribution engine and business need, the SOA technology was applied as a bridge between them. For this purpose, each application function of the existing software distribution engine needed to be transformed into a service or services.

SOA technology allows service assemblers to place multiple distribution services into processes. After the distribution related processes are ready, a series of steps is executed by invoking distribution services.

Once an operator of the SDMS system submitted a software distribution related request (create software, delivery software etc.) via an operation management portal, this system will invoke corresponding processes which including services extended from a software distribution engine. If the software distribution engine receives a distribution request by the service, it will create a task with the target device information and store the jobs in the task repository.

These tasks are completed by an operator at the setting level while the actual work of software distribution is enforced by the distribution agent and the software distribution engine. As soon as the two modules are connected, the distribution agent queries the engine for work. The distribution agent then verifies that the device is enrolled, before checking its task repository for any software submitted for distribution. For a distribution task, a URL address that points to a location where the new software can be obtained is retrieved from the task repository. The software is then distributed to the home gateway with the help of the distribution agent.

5. Development Roles and Tools

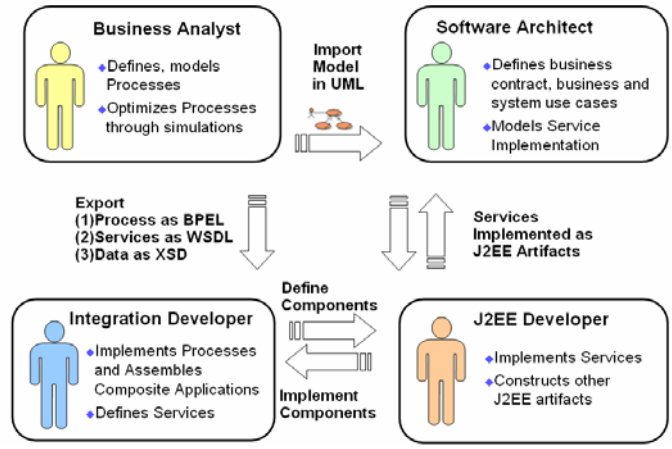

Figure 4. The relationships between SOA roles

The development of the Software Deployment solution is a team effort which involves Business Analysts, Integration Specialists, Software Architects, and J2EE Developers. The relationship between those roles is illustrated in Figure 4.

(1) Business Analysts

Business analysts are responsible for the software deployment processes and business design, in home businesses. Furthermore, they also use modeling tools to document software deployment, define business measures related to the goals of the solution, and simulate the process to verify the process will meet the goals set for it.

(2) Software Architect

The Software Architect is responsible for translating the business analysts design into a set of software component design specifications for implementing the service definitions and business objects identified by the analyst.

There are inevitably various constraints placed on the overall design and its implementation. This software architect contributes to the development of the system's

software architecture by making key technical decisions that deal with these overall design challenges

Specifically, the SOA Software Architect performs the following tasks: (1) identify design elements, (2) identify design mechanisms, (3) incorporate design elements, and (4) identify services.

(3) Integration Developer

The Integration Developer is responsible for integrating services into the business process definition. This specialist typically uses visual composition tools to wire abstract service components that comprise the business processes.

(4) J2EE Developer

The J2EE Developer is responsible for implementing the design for service provided by the software architect.

6. Software Engineering for SOA

6.1 RUP for SOA

The RUP for SOA plug-in provides extensions to RUP specifically for use in SOA. Software Engineering for SOA employs business models both in understanding the functional requirements of the project, and the services required to support those requirements. It describes the connection between business modeling and the system model. This methodology also creates a roadmap through the RUP life cycle when developing service-oriented solutions with the following extensions.

Inception phase activities

During this phase a number of steps are undertaken. First, a project manager evaluates the scope of the business need and identifies a schedule for task completion. Second, a business analyst is required to identify business goals and business processes.

Next a software architect assesses the technical requirements of the project. This analysis results in the identification of each of the phases required for the development of service package that will meet the business need.

The software architect also ensures that each of the required services is developed independently. If the services are not built autonomously, they may develop dependencies that will limit their reusability.

Elaboration phase activities

The elaboration phase focuses on the technical and nonfunctional requirements of the project. This phase also requires the completion of several steps.

First, it requires the identification of the various constraints imposed upon the planned system. Then, the resulting analysis proposes a service-oriented architectural style and uses its

technical requirements to define a detailed plan. This includes initial partitioning of the system into a set of service components and software modules.

In the tasks of design services, an initial service model should be identified, and the operations of candidate services should be specified with message input and output requirements.

In the later stages of the elaboration phase, the design of the service is further refined, and subsystems within the service are identified. This provides insights into the expected behavior of the service.

Finally, the elaboration phase conducts a testing of tasks, which focuses on validating the individual services. Consequently, the focus in elaboration is on ensuring that the service encapsulates its data requirements completely.

Construction phase activities

The focus in construction is on analyzing the business-use cases, and business process models, which are extracted from the design document. This phase also evaluates the service model, produced by the software architect in the elaboration phase. The purpose of this analysis is to identify appropriate services and service collaborations that realize the projects goals.

In this phase, the tasks are extended to look specifically for services and mechanisms appropriate to the overall SOA. This results in a transformation of service models into 'system use cases'. This transformation must occur before the generation of either an implementation model or programming language artifacts that represent service implementation

In doing this the phase conducts a component implementation step. In this step components are used as the implementation artifacts for services. From these artifacts, most design tools can generate client assessors for the specified services.

6.2 Model Driven Development for SOA

Model-driven design is the process of describing a system as a series of graphical models prior to implementation. Clearly, modeling is a critical first step in the creation of an SOA because it helps both IT and business leaders gain a better understanding of the impact of their roles on their organization and the value their collaborative efforts will deliver through the SOA.

A model-driven approach for service-oriented software development encourages early design testing and continuous formal analysis of quality of service. This includes an approach to model transformations based on graph transformation systems, a model-driven development process for service-oriented computing, tools supporting model transformations, and iterative analysis

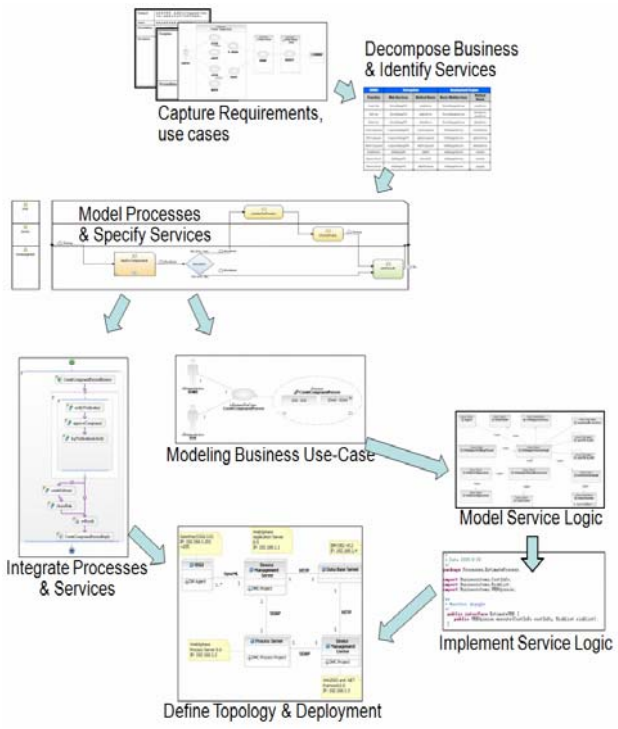

Figure 5. Model-driven design of the SDMS system

The Service-Oriented modeling process is comprised of three major steps: identification, specification and realization. The relationship between those steps is illustrated in Figure 5. The most important role of identification activities is the production of the service model. The service model is comprised of candidate services that, ultimately, support business services, processes and goals of the enterprise.

The second, service specification, is tasked with identifying and specifying components that will be required to realize services. The final step, realization, is a key architectural step that involves the exploration of alternatives to realize the implementation of services.

7. Implementation

To facilitate the adoption of SOA, the project establishes this distribution management system in phases. This system will focus on the service management process. This phase includes exposing existing distribution management functionality as services to other home community applications.

7.1. Defining business requirements

Some distribution management engines have already been developed and have the ability to deliver applications to destination environments.

However, while these engines can provide basic functions, they cannot meet complex business needs. For instance, an intelligent home domain application could conceivably need more application services than a basic software distribution engine can provide.

This study scenario consisted of two different functional requirements: First, in order to increase levels of customer satisfaction, the system needed to provide an automatic delivery mechanism, which is able to deploy the latest application.

Second, user management of information regarding other subsystems was to be a key function. This is why the SDMS system requires the users to login with a user identity and password.

In addition to these two primary functional requirements, the project also dealt with the non-functional requirement of reusing existing software distribution application functionality as Web services. This addressed the problem of heterogeneous system integration, and allowed other applications to consume these services.

The development infrastructure maintains the software distribution system, by using tools for saving time and avoiding redundant work. Those tools assist business analyst in detecting, tracking, and designing business requirements.

7.2. Service identification

This project initially uses a top-down approach starting with business modeling. This allows the users to make progress on business models iteratively and incrementally.

Typically, migration to an SOA solution involves integration of existing systems by decomposing them into services, operations, business processes and rules. The domain scenario of the home SDMS can further decompose the functional areas into processes, sub-processes and high-level business use-cases as depicted in Figure 6.

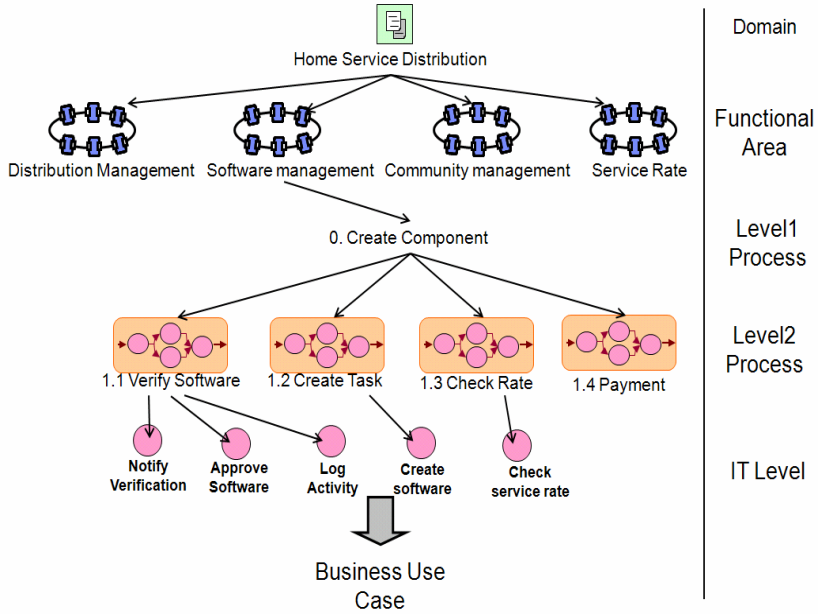

Figure 6. Home Service Distribution Management Domain Decomposition

For the purposes of our scenario, the 'Create Component' stage can be decomposed into at least five business use-cases including: 'notify verification', 'approve software', 'log activity', 'create software', and 'check service rate'. From this approach, the preliminary services also have been identified.

Once business models have been optimized, the model is translated to UML2 by architect design tools. UML2 transforms the business requirements into software system requirements and models. All this ensures that SDMS implementation is driven by business requirements.

The claim information class (Figure 5) has the same attributes it had in the business process model. A signal from UML2 will also transfer the information found in the original business model diagram into the business contract specifications, which are represented as «BusinessEntity» classes.

One step of service identification is an analysis of existing assets, in order to maximize the reuse of the software distribution engine. When performing an existing distribution engine analysis, a bottom-up approach is applied to identify candidate services. During this activity, technical constraints related to existing systems are also considered. The use of this approach assists in identification of potential risks.

After the candidate services have been identified, the services are categorized into a service hierarchy as illustrated in Table 1. The categorization reflects the composite nature of services and helps to determine composition and layering. Categorizations are typically identified through functional area analysis, during domain decomposition activities.

Table 1. Distribution Service identification

SDMS	Delegation		Deployment Engine	
Function	Web Services	Method Name	Basic WebServices	Method Name
Create User	DeviceManageWS	createDevice	DeviceManagerServices	createDevice
Edit User	DeviceManageWS	updateDevice	DeviceManagerServices	deleteDevice ↓ createDevice
Delete User	DeviceManageWS	deleteDevice	DeviceManagerServices	deleteDevice
Create component	ComponentManageWS	createComponent	SoftManagerServices	createSoftware
Edit Component	ComponentManageWS	updateComponent	SoftManagerServices	updateSoftware
Delete Component	ComponentManageWS	deleteComponent	SoftManagerServices	deleteSoftware
Install Service	SoftManageWS	addSoft	JobManagerServices	createJob
Remove Service	SoftManageWS	removeSoft	JobManagerServices	createJob
Delivery Service	SoftManageWS	addSoftToGroup	JobManagerServices	createJob

7.3. Design Service Architecture

At previous stages during the implementation process, the functional requirements of the generic use cases have been identified for home service distribution. The generic-use cases are then used as criteria to select the appropriate SOA scenario containing software capable of fulfilling case requirements.

In this stage, the software architect will import business processes produced through previous work and refine their application designs. Patterns for e-business can also be applied in the system to accelerate the design of the solution architecture. Application patterns are used to provide a conceptual layout of the application components and data within a business pattern or Integration pattern interaction.

As business patterns are applied to this system architecture, the services exposed form distribution engine applications also adhere to sets of non-functional requirements that differ from the requirements for the original applications. Using software patterns to satisfy non-functional requirements helps the developer and architect build an architecturally consistent service that complies with software development best practices and principles.

In choosing a service design pattern, there are some options that can be considered: WS Response Template Pattern, Requester Side Caching Pattern, and Aspects Patterns.

7.4. Deploy the composite application

Before assembling software distribution processes, the services need to be constructed to meet the distribution requirements with the desired nonfunctional characteristics. The core of service application can be further divided into a controller layer, and an entity layer. The developer can implement those distribution services by leveraging Enterprise JavaBean patterns in the controller layer, such as the session facade, message facade, and the business delegate.

During the Service flow specification phase, an assembly process editor can provide process implementation via component assembly. To do this both skeleton assemblies and data definitions are imported from Business Modeler.

In the proposed system, IBM WebSphere Integration Developer was adopted. The development tools used in the system support standard Business Process Execution Language (BPEL) [17], which has the capability of deploying process definitions to any BPEL-compliant, application server. A BPEL process is a flow that describes the order and conditions in which services are invoked.

BPEL provides a means of connecting web services and specifying how collections of web services can be jointly used to provide more complex functionality in the form of a business process. It also provides a model and syntax for describing the behavior of a business process in terms of the interactions between the process and its partners. Each interaction with a partner occurs through web service interfaces, and the BPEL process includes the state and logic that coordinate service interactions to achieve distribution management domain goal.

For high-level distribution process designs, a BPEL process implements operations through verifying software or check rate activity. These operations are available in the external interface of the process. A business process can be invoked as an SCA component through that interface. Existent business process interfaces, for ‘receive’ and ‘reply’ activities, are available to SCA through export. Non-SCA components are also available on a stand-alone basis.

8. Result

The project was used to prove that a SOA-based SDMS implemented by software engineering methodology could be applied to device management. In the SDMS, the operation management portal provides administration pages that have all been displayed using the interfaces illustrated in figure 7.

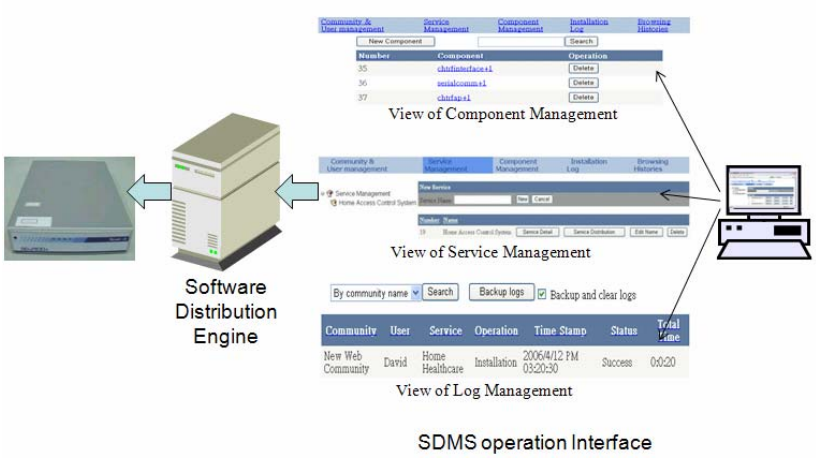

Figure 7. The views of the administration pages

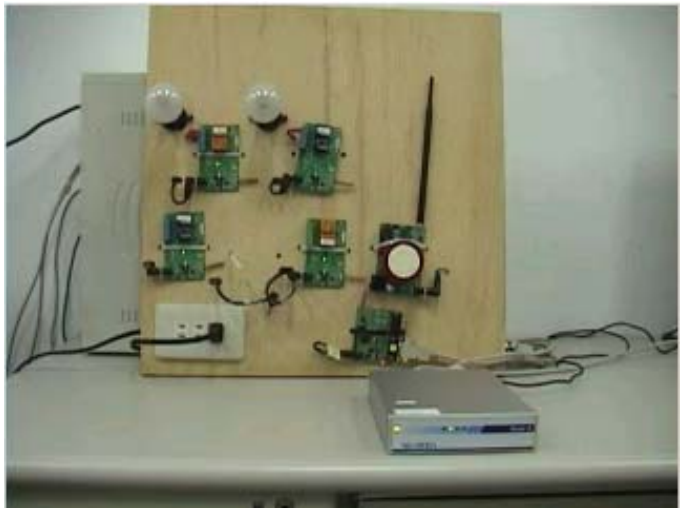

Figure8. Home control applications form SDMS

Furthermore, in this project home application were successfully deployed to a home gateway. Figure 8 presents the actual result of the home control applications delivered form the SDMS system

9. Monitoring and Evaluation

Ongoing work is currently investigating the practical aspects of building a SDMS using SOA technology. A software engineering process is applied to perform development guidance with a SOSE methodology. However, this work is only a part of the overall process management. Successful Business Process Management (BPM) in SOA implementations also relies on well-defined key performance indicators (KPIs) defined in a Business Measures Model.

A KPI is a performance meter for a specific business activity that has an associated business goal or threshold. The tracking and analysis of KPIs helps business users gain the insight required for business performance optimization.

This project utilizes a Business Measures Editor (BME) to describe the business performance aspects of a business model as a Business Performance Model (BPM). This model is used by WebSphere Business Monitor to create custom dashboards for intelligent home service providers. The dashboards in turn follow the status of the software distribution process, examine and analyze historical information, and even perform trend analysis. When exported this model provides a monitoring tool that can answer several pertinent questions:

1. Which software distribution process log events should be gathered?
2. What are the business performance measures for this software distribution process?
3. What event and measurement values best represent a specific business situation?
4. How are emerging situations detected from among the monitored processing events?

Using this information, this monitor performs the following actions. First, it receives process execution events from event repository. Second, it will filter events specified by the business measures model. Third, it will use the model to calculate and update business measures based on the input from the filtered events. Finally, it will use the model to detect business situations. When detected, a situation event will be submitted to event repository for further situation handling.

This project uses three kinds of dashboards for observing the information of the distribution process. The ‘Active Instances’ view shows the details of a distribution process at runtime. The ‘Reports’ view displays performance reports about historical values of aggregate business measures relative to a time axis. Finally, the ‘Dimensions’ view permits users to generate multidimensional reports that analyze different business aspects of historical business-performance data.

10. Conclusion

This paper, presents a mechanism to apply SOA architectures for software distribution management in smart living environments.

This research has successfully demonstrated a software distribution management system using Service-Oriented Architecture. The SDMS not only makes home application delivery more effective, but also reduces the gap between the basic software distribution engine and smart-home domain needs

This use of the SDMS system applied reused distribution functions to a complex domain. In doing so, the SOA technology was found to be very effective in addressing traditional

reuse problems. For an existing software distribution engine, it is advisable to have a service set that can guarantee a higher degree of support.

The RUP Plug-In for SOA provides guidance for the Software Architect and Designer in developing artifacts that can be used to supplement implementation activities for the SOA based SDMS system.

In terms of service-oriented modeling, the model driven development for SOA is a very prescriptive method that includes several complementary approaches to identifying services for home application distribution domains.

11. Acknowledgement

This research was supported by the Ministry of Economic Affairs, Taiwan, and R.O.C. under grant 94-EC-17-A-02-S1-049.

12. Reference

[1] Dillinger, M., Becher, R., "Decentralized software distribution for SDR terminals" , *IEEE Wireless Communications*, 9(2), 20 – 25, 2002.
[2] McConnell, S., "Annualized software delivery", *IEEE Software* , 14(1), 103-104, 1997.
[3] Hudepohl, J.P., "Measurement of software service quality for large telecommunications systems", *Selected Areas in Communications, IEEE Journal on*, 8(2), 210 – 218, 1990
[4] Xianjun Geng, Gopal, R.D., Ramesh, R., Whinston, A.B., "Scaling Web services with capacity provision networks" , *Computer* ,36(11), 64 – 72, 2003
[5] Briskin, G.; Clements, M.; Cutts, D.; Kakaz, K.; Mattingly, S.; Watts, G. , "Control of large scale distributed DAQ/trigger systems in the networked PC era", Nuclear Science Symposium Conference Record, 2000 IEEE, 15-20 Oct. 2000
[6] Andersson, J, "A distribution system for pervasive computing" , Software Maintenance, 2000. Proceedings. International Conference on , 11-14 Oct. 2000
[7] Topalis, E.; Mandalos, L.; Koubias, S.; Papadopoulos, G.; Nikiforakis, I., "A novel architecture for remote home automation e-services on an OSGi platform via high-speed Internet connection ensuring QoS support by using RSVP technology", *Consumer Electronics, IEEE Transactions on*, 48(4), 825 – 833, 2002
[8] Gold, N.; Mohan, A.; Knight, C.; Munro, M., "Understanding service-oriented software", *Software, IEEE*, 21(2), 71 – 77, 2004
[9] Pasley, J., "How BPEL and SOA are changing Web services development", *Internet Computing, IEEE* , 9(3), 60 – 67, 2005
[10] Bichier, M.; Lin, K.-J., "Service-oriented computing", *Computer*, 39(3), 99 – 101, 2006
[11] Karhunen, H.; Jantti, M.; Eerola, A., " Service-oriented software engineering (SOSE) framework", Services Systems and Services Management, 2005. Proceedings of ICSSSM '05. 2005 International Conference on, 13-15 June 2005
[12] Ketabci, M., "Evaluation of OOAD methodologies", Computer Software and Applications Conference, 1991. COMPSAC '91., Proceedings of the Fifteenth Annual International, 11-13 Sept. 1991
[13] Hong, S.; van den Goor, G.; Brinkkemper, S., " A formal approach to the comparison of object-oriented analysis and design methodologies", System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on, 5-8 Jan. 1993
[14] Lange, C.F.J.; Chaudron, M.R.V.; Muskens, J., "In practice: UML software architecture and design description", *Software, IEEE*, 23(2), 2006
[15] Peltz, C., "Web services orchestration and choreography", *Computer*, 36(10), 46 – 52, 2003
[16] Min Luo; Goldshlager, B.; Liang-Jie Zhang, "Designing and implementing Enterprise Service Bus (ESB) and

SOA solutions", Services Computing, 2005 IEEE International Conference on, 11-15 July 2005
[17] Pasley, J., "How BPEL and SOA are changing Web services development", *Internet Computing*, IEEE, 9(3), 60 – 67, 2005

Authors

Ing-Yi Chen received a B.S. in physics from National Central University, Taiwan, in 1984, and M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Arizona, USA, in 1989 and 1992, respectively. Dr. Chen is currently an associate professor in Computer Science and Information Engineering Department, National Taipei University of Technology, Taipei, Taiwan. Prior to joining NTUT, he served as chief technology officer for Chinatimes from 2000 to 2001 with responsibility for the corporation's strategic technology planning and external technical affiliations with universities. Previously, he was a faculty member in the Department of Electronic Engineering at Chung Yuan Christian University, Taiwan between 1992 and 2000. His research interests include various topics in ubiquitous computing and web services. A member of the Phi Tau Phi scholastic honor society, Dr. Chen was a recipient of the IEEE/ACM Design Automation Scholarship Award in 1991, and the Distinguished Teaching Award at CYCU in 1996.

Chao-Chi Huang received the B.S. degree in Electrical Engineering from the Chung Yuan Christian University, Chung Li, Taiwan, R.O.C., in 2001, and the M.S. degrees in Electronic Engineering from the Chung Yuan Christian University, Chung Li, Taiwan, R.O.C., in 2003. Currently, he is working toward the Ph.D. degree at the National Taipei University of Technology, Taipei. His research interests include Single Sign-On systems, and Service-Oriented software Engineering.