A Service-based Approach to Designing Cyber Physical Systems

Hyun Jung La and Soo Dong Kim
Department of Computer Science
Soongsil University
511 Sangdo-Dong, Dongjak-Ku, Seoul, Korea 156-743
hjla@otlab.ssu.ac.kr sdkim777@gmail.com

***Abstract*— A Cyber-Physical System (CPS) is defined as integrations of computation and physical processes. In CPS, downsized embedded devices monitor and manage the physical process. Mobile Internet Device (MID), as a portable handheld device designed for mobility providing a down-graded computing capability, can be used as a strong candidate for client-side devices in CPS. Since these physical devices have limited resources, it is not possible to execute complex computation and processes. To cope with this challenge, we apply Service-oriented Architecture (SOA) or Cloud Computing (CC) concepts to CPS, called a service-based CPS. To realize a service-based CPS, we first define overall architecture with three tiers. And, we present key methods to design each tier in the architecture. The design methods are defined to deal with design challenges of CPSs such as dynamic composition, dynamic adaptation, and high confidence CPS management. Then, we perform a case study to show applicability of our approach. With this approach, we hope that CPS can even handle complex and resource-consuming physical processes with a downsized MID.**

Keywords-cyber physical system, CPS, service-based approach, design method

I. INTRODUCTION

A CPS is defined as integration of computation and physical processes. In CPS, downsized and embedded devices execute physical processes by monitoring and controlling entities in the physical world. Unlike embedded systems, physical devices can expose their functionality to outside with networking capability. In addition to different kinds of sensors and actuators, a MID, which is a portable handheld device with a down-graded computing capability, can be used as a kind of physical devices in CPS. The devices normally have unique features such as *limited resource, a strong support for Internet access,* and *location sensitivity*. The feature of *limited resource* forbids deploying and running complex computation and processes on physical devices in CPS. This is the first motivation for our research.

Since CPS is quite new computing system, it is not straightforward to develop CPS with conventional design approaches. Most of the works on CPS raise design challenges or design issues such as dynamic composition, and high confidence software design [1][2][3]. However, solutions for only a few challenges have been proposed. Moreover, although CPS is a multi-disciplined research area, viewpoints of the proposed solutions seem to be confined to embedded system or network area. Therefore, the solutions need to be rethought with the consideration of software engineering. This is the second motivation of our research.

To deal with the intrinsic challenges of CPS, we apply SOA[5] or CC[6] concepts to CPS. In SOA and CC, service providers publish reusable services and service consumers reuse the appropriate services. With the concepts, service consumers can be provided with their required functionality with a minimum of computing resources.

Conventional CPS typically has employed two-tier architecture; one for a number of physical devices in a grid and the other for a control system which determines and provides necessary functionality [4]. The service-based CPS proposed in our work is defined with three tiers; *Environmental Tier* for the target physical environment, *Control Tier* for making decisions for networked physical devices, and *Service Tier* for managing reusable services. With the service-based CPS, we can deal with the following design issues.

- Dynamic composition thanks to standardized interface and loosely-coupling characteristic of services
- Hardware heterogeneity thanks to programming-neutral characteristic of services

Moreover, we can utilize research achievements of autonomous service management, which have been investigated in SOA areas, to cope with dynamic adaptation.

In this paper, we present a service-based approach to design CPS. In section 2, we present a survey of related works. In section 3, we define an architecture or service-based CPS in terms of the three tiers. In section 4, we present key methods to design CPS by considering three tiers. And, we performed a case study, *Safety Guard*, to show an applicability of our approach.

II. RELATED WORKS

Prasad and his colleague present a complete design suite for embedded CPSs by considering complex interactions between communication, computing, and physical components [2]. In the suite, they emphasize network and node level modeling, present various types of analyses including reliability, security, performance, and various QoS metrics, and provide facilities to model device mobility and represent the dynamism of mobile nodes in the system model. Overall design process is needed to present and detailed design guidelines are required.

Lee examines the challenges in designing CPS by considering whether current computing and networking

978-0-7695-4147-1/10 $26.00 © 2010 IEEE
DOI 10.1109/ICIS.2010.73

technologies can support CPS design [3]. First, this work identifies key requirements for CPS and addresses a demand for rebuilding computing and networking. This work only considers building core abstractions which are unique to CPS, rather than considering various design issues.

Abdelwahed and his colleagues present an approach to designing high confidence software for CPS [7]. They first CPS-intrinsic characteristics such as keeping complexity and scale, managing dynamic and uncertain environment, configuring computing models on demand, and ensuring high confidence. Then, they present four design methods; model-driven system execution modeling, model-based diagnosis, online control, fault-adaptive control, and trustworthy middleware infrastructure. And, they present an architecture for CPS. This work only considers designing control layer and needs to present more detailed design guidelines for the given methods.

Kansai and his colleague introduce a model-integrated development approach by covering all aspects of the hardware and software components [8]. They present two distinguish models; *Computation System Model* reflecting model world and *Computational Model* reflecting real world. And, they raised an issue on integrating these separate models to a CPS model. To handle this issue, they propose a continuous integration process which is tailored to CPS. This work only considers a way to integrate two levels of models, rather than covering wide range of develop process.

Most of the works address key design issues to consider and their potential solutions. However, solutions of the only limited set of design issues are proposed such as model integration and abstraction. In addition, the proposed solutions need to be presented in more detail.

III. ARCHITECTURE OF SERVICE-BASED CYBER PHYSICAL SYSTEMS

We identify two motivations behind the research on service-based CPS. One motivation is about the availability of mobile network such as WiFi and 3G. Utilizing mobile internet, deployment of CPS which requires a good level of network connectivity becomes feasible and widely available

The other motivation comes from the fact that services technology such as SOA and CC can resolve the problem of limited resources on physical devices. Hence, providing the needed CPS functionality through services solves the resource constraint problem.

There is a general consensus on what CPS is, what it can do, and how it can be used. However, there is a lack of consensus on key elements of CPS, their relationships, and communication models of CPS. Hence, we define service-based architectures of CPS. In defining the architecture, we consider the following observations and assumptions on CPS.

- Physical devices are connected over network to the control system which performs key computations.
- Software functionality is not tightly coupled to hardware elements.
- CPS requires real-time and on-demand processing.

Based on these, we define the architecture of service-based CPS in Figure 1. Our service-based CPS consists of three tiers; *Environmental Tier, Control Tier,* and *Service Tier*.

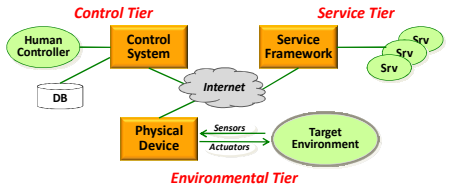

Figure 1. 3-Tiers of Service-based CPS

Environmental Tier consists of physical devices (typically in a form of embedded device) and a target environment which includes end-users using the devices and their associated physical environment.

Service Tier is a typical computing environment with services in SOA and CC. In that, a number of services are deployed on *Service Repositories* and a *Service Framework* manages the services and interacts with service consumers.

Control Tier is to receive monitored data which are gathered through sensors, to make controlling decisions, to find right services by consulting *Service Framework*, and to let the services invoked on the *Physical Device*.

Hence, the architecture of service-based CPS is different from the conventional two-tier CPS where required functionality is hard-bound to *Control Tier* [4]. The key benefits of service-based CPS can be summarized in three folds;

- Being able to reuse generic services for multiple CPSs by decoupling *Control Layer* from *Service Layer*
- Being able to adapt services for the monitored contexts
- Being able to handling functional changes by separating *services* from *physical devices*

IV. DESIGN METHODS

Since different tiers have different essential design considerations or challenges, we need to reflect all the concerns of the tiers to CPS design.

Figure 2 shows an overall process to design CPS. The first three phases of the process are applied to all the tiers. From the fourth phase, we have three branches to consider all the three tiers separately.

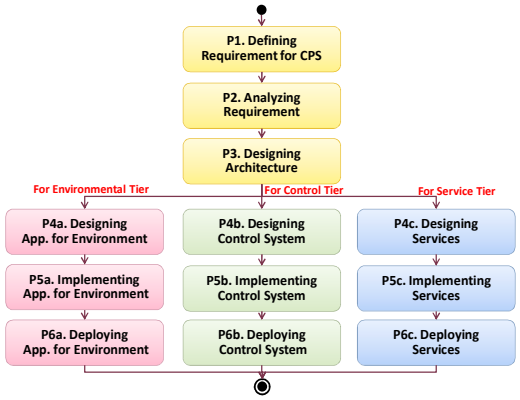

Figure 2. Overall Process

In this paper, we only focus on defining design methods without considering implementation and deployment.

A. Designing Environmental Tier

An *environmental tier* consists of multiple heterogeneous sensors and actuators. All the sensors and actuators deploy applications which handle themselves. The applications perform the following capabilities;

- To provide all the information gathered from users and their environment, which is performed by sensors
- To transfer the gathered information to applications in the control tier
- To communicate with applications in the control tier by using their own network capability
- To deliver results to users, which is performed by actuators

Since sensors and actuators have few or limited resources, they cannot deploy too complex applications. Hence, when designing these applications, we should make right decision on assigning functionalities to *Environmental tier* or *Service tier*. Therefore, based on the functionality model (i.e. use case model) produced in the previous phase, be sure that applications in this tier only perform the following functionalities;

- Functionality consuming a few computing resources
- Functionality requiring strict performance constraints
- Functionality handling privacy-related information

B. Designing Control Tier

Among three tiers, a *control tier* plays an important to role in realizing service-based CPS. Most design challenges identified in previous works such as [1], [2], and [3] are more related to this tier. By investigating previous works, we identify the essential roles of control systems as followings;

- Monitoring physical components as well as services effectively and efficiently
- Determining the most appropriate service and composing the services at runtime
- Managing physical components as well as services in an autonomous way

1) Monitoring Physical Components and Services

Monitoring physical components and services spends many efforts so to have a bad influence on the performance. Hence, we present an overall structure to effectively monitor physical components and services as shown in Figure 3.

Like SOA, searching service registries located remotely at runtime is quite inefficient and so not feasible in practice. Hence, we utilize a kind of cache for the published services, called *Local Registry*. *Local Registry* maintains the comprehensive information about published services relevant to the domain or organization in terms of functionality, QoS, and context information. And, *Local Registry* manages information about all the physical devices including types of sensors and actuators, their locations, and other device-related information.

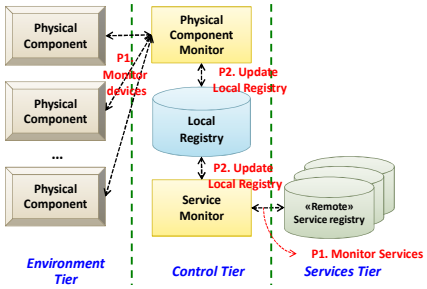

Figure 3. Supporting Effective Monitoring

Service Monitor checks remote service registries whether new services are added or not and updates the description of services in *Local Registry* with up-to-date information by pro-actively monitoring services. Likewise, *Physical Component Monitor* monitors all the physical components under the given CPS and update the up-to-date information.

To support effective monitoring, *Physical Component Monitor* and *Service Monitor* are designed by using *pulling* pattern or *pushing* pattern. With pulling, *monitors* invoke monitoring methods to service registries, published services, and physical devices. With pushing, *monitors* implements *publish-and-subscribe* pattern to register and to get notified from service registries, published services, and physical components. Acquiring QoS values with *pushing* pattern is efficient, but it requires the active collaboration of service providers. Figure 4 shows a scenario using *pushing* pattern. The listener (i.e. *Service Monitor)* is registered to a published service, update on QoS values is notified, new QoS values are retrieved, and the values are updated to *Local Registry*.

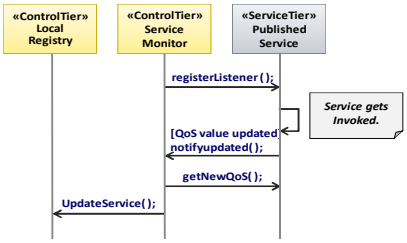

Figure 4. Acquiring Updated QoS values using *Pushing* pattern

2) Ensuring Dynamic Composability

In conventional CPS, physical components are equipped with functional and network capabilities so that, ideally, they are dynamically composed for current contexts or situations. On the other hand, in service-based CPS, physical components are in charge of sensors and actuators, and context -aware functionalities are located on *services tier* [9]. Hence, in our approach, both services and physical components belong to targets which are dynamically composed.

To support dynamic composability of physical components and services, monitoring should be preceded. As shown in Figure 5, we define *Service Composer*, *Physical Component Configurator*, and *Interface Adapter*. Upon service invocation from control systems, *Service Composer* listens to the invocation, selects a most suitable service by looking up *Local Service Registry*, invokes it, and sends service results to *Physical Component Configurator*.

Physical Component Configurator determines the most appropriate physical components by using the monitored results and forwards results to end users. *Interface Adapter*, which will be discussed in the following section, is to resolve the interface incompatibility between callers and callees. This component is needed since all the interfaces of the determined services are not matched to the ones known by applications in environmental tier.

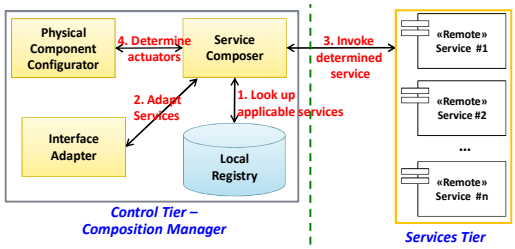

Figure 5. Supporing Dynamic Composability

At runtime, upon getting service invocation, *Service Composer* finds the candidate services by comparing monitored context information and service request with information stored in *Local Service Registry* (step 1), adapt service interfaces if needed (step 2), invokes the determined service (step 3), and send results to *Physical Component Configurator* (step 4).

3) *Ensuring Service Adaptability*

Components in component-based development (CBD) can be tailored in two ways; by using some forms of *adapters* and by using *variability* mechanism. These ways can also be applied in service-based CPS.

Control systems are treated as service consumer in service-based CPS. In SOA, it is hard to find services which can fully meet consumers' expectation in terms of service interfaces and service functionality. That is, there exist *mismatches*. To increase service reusability, it is evitable to adapt services without directly modifying the service. Hence, several kinds of *adapters* need to be designed to resolve the mismatches. The representative form of these adapters is an *Interface Adapter*.

Applications in the environmental tier have predefined interfaces for service invocations, and service interfaces cannot be matched to those interfaces. Hence, *Interface Adapter* is to mediate mismatches between those two interfaces. Typical mismatches on interfaces are mismatch on the names of operations, mismatch on the datatypes of parameters and return value, mismatch on the orders of parameters, and mismatch on value ranges of parameters.

First, human administrators decide types of mismatches, define mapping rules for those mismatches in a machine-readable form, and store the rules in DB. Then, they design this adapter by applying *Adapter pattern* [10] without directly modifying services. Figure 6 shows how *Interface Adapters* works at runtime to resolve interface mismatches.

Applications deployed on the end user's physical devices have a prediction on service interface to invoke, *op1(paramList1)*. *adaptedClass* plays a role of resolving interface mismatches by overriding a method of the provided service. When an interface mismatch occurs, *adaptedClass* first looks up suitable mapping rules from DB, adapt the interface, and invoke the service with adapted interface.

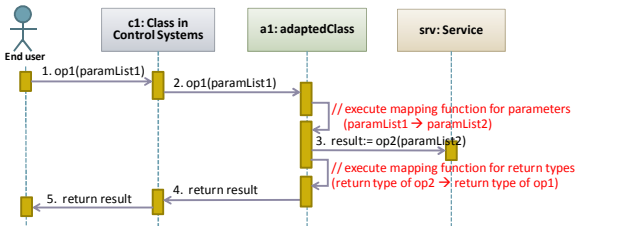

Figure 6. Dynamic Model of *Interface Adapter*

Another way to adapt services is to utilize *variability* embedded in service design. A service inherently is designed with common functionalities and minor differences in the functionalities, which will be discussed in a section C. Hence, it is possible to customize the service by selecting a variant for the given variation point. When a service is firstly invoked by control systems, the control systems decide a right variant by using selection and plug-in methods defined in a *Required Interface* [17]. Once a variant is selected, the variant should be managed in DB so that the variant is automatically chosen whenever the service is invoked.

4) *Ensuring Autonomous CPS Management*

In CPS environment, there is high possibility to change environments dynamically. This leads a problem, which is that the functionality cannot be executed or provided to end users. Hence, control systems should be designed with the consideration of autonomous management.

To manage service-based CPS in an autonomic manner, we introduce three conceptual elements; *fault*, *cause*, and *dynamic adapter* [12]. A *fault* is an observed state of a target element which reveals an abnormality. A *cause* is defined as a reason which results in the occurrence of some fault. A *dynamic adapter* is a means to handle a fault by healing its associated cause.

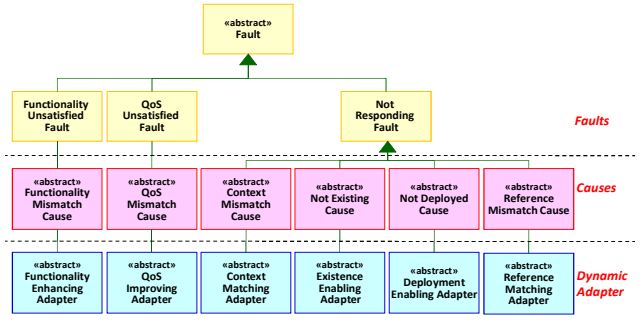

Figure 7. Hierarchy of Faults, Cause, and Dynamic Adapter for Service-based CPS

There can be many different types of faults and their causes in service-based CPS, and they have not been identified. Without such taxonomy, it is challenging to derive a complete set of dynamic adapters. Figure 7 shows a hierarchy of faults, causes, and dynamic adapters which are derived from characteristics of traditional CPS.

With this hierarchy, we present an architecture for autonomous fault management as shown in Figure 8. *Fault*

Identifier identifies the faults by comparing monitored data to normal behavior description [13]. *Cause Detector* determines the most plausible cause by consulting reasoning knowledgebase. *Adapter Determiner* selects and invokes the most appropriate dynamic adapters to remedy the known cause(s) by using the hierarchy in Figure 7.

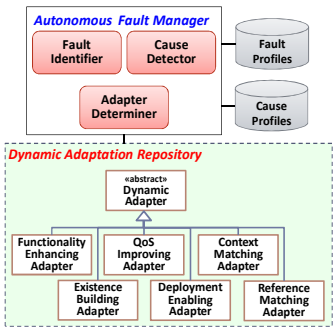

Figure 8. Supporing Autonomous CPS Management

Unlike traditional SOA, physical components as well as services are management targets. Note that, there is an adapter for resolving context mismatches between physical components and published services, which is *Context Matching Adapter*. Except for this adapter, all the adapters are much similar to ones proposed in [11]. The only difference is that control system should consider physical components as a management target. Accordingly, detailed algorithms in [11] need to be slightly modified.

C. Designing Service Tier

The services to invoke are determined by control systems in the control tier, and results are forwarded to the end users through the control tier. The service is a modular web service that provides reusable and cohesive functionality. It can be SOA services [5] or cloud services [6].

Design instructions of this phase are similar to conventional SOA service design. Since the services are shared by multiple users, the services should be design by considering the commonality [14] and adaptability [15].

First, we give guidelines for designing services with high commonality. Like CBD and SOA, we start with identifying common functional items by considering multiple potential end users [14]. For all functional items (i.e. use cases) in the functional model, we evaluate the degree of commonality. And, we consider other criteria such as cohesiveness, coupling, marketability, and other domain-specific criteria to figure out a unit of service.

After determining services, services are designed in terms of internal and external designs. Being design internals of the services is to model structural and dynamic model, which is much similar to object-oriented designs, except for designing variabilities [16].

Since services are exposed to multiple control systems, the control systems invoke the services through interface. Hence, the services interfaces (i.e. *provided interface*) are designed with a standardized form such as WSDL. In addition to information on WSDL, service interfaces for CPS should include provide *contexts* to be invoked, *QoS*, and other constraints. That's why service invocation is largely determined by context as well as required functionality.

Next, we present guidelines for embodying adaptability to common services. Since the services are provided with black-box forms, it is hard for control systems to modify the services even if there are some needs to customize the services. To be able to adapt services more flexibly, services should be designed by considering variabilities. A common mechanism to support variability/adaptability is to define *Required Interface* to select one variant for a variation point or accept plug-in objects from control systems [17]. Figure 9 shows design of the service by realizing adaptability. The *required interface* let control systems plug in an object (i.e. *MyPlugIn*) reflecting end user-specific functionality.

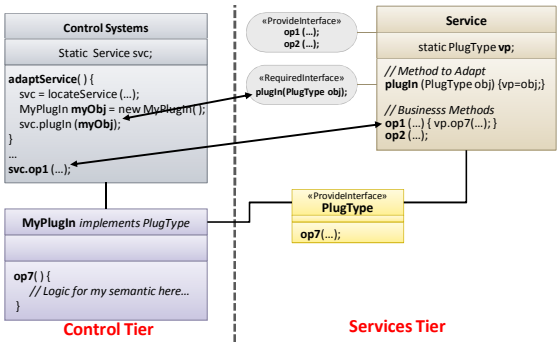

Figure 9. Required Interface and Plug-in Mechanism

V. Case Study

We perform a case study of a target domain, *Safety Guard*. There are two kinds of end users; *children* and *elderly people*. According to the contexts determined by a set of monitored data, the control system determines services to invoke; *Children Keeper Service* which notifies emergency case to parents or police offices and *Health Care Service* provides online treatment functionalities.

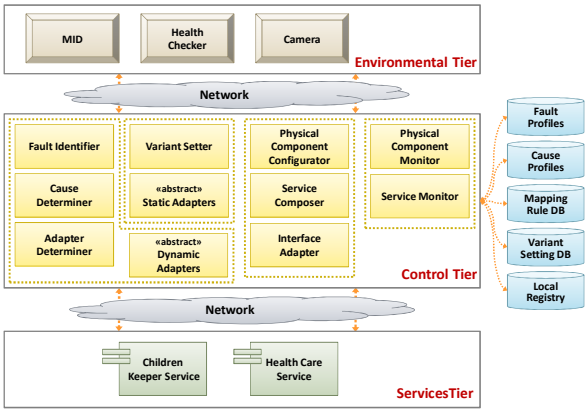

Figure 10. Architecture of *Safety Guard*

Figure 10 shows overall architecture for *Safety Guard*. Components in the control tier seem to be generic to any types of service-based CPS, but the detailed algorithm should be specialized to *Safety Guard*. Note that there are several kinds of DBs that help to invoke services such as *Mapping Rule DB* for storing monitored data and its

corresponding services and *Variant Setting DB* for storing variants for end users. Table I. shows the fragment of rules for invoking services according to the monitored data, which is stored in *Mapping Rule DB*.

TABLE I. PART OF THE RULES FOR INVOKING SERVICES

User Type	Place (Location)	Temperature	Heartbeat	Service
Children	within 1km from safety area	-	-	*AlarmParent()*
Children	In dangerous zone			*callParent()* & *callTeacher()*
…	…	…	…	…

If the type of user is children and current location is detected as inside dangerous zones (which are defined by parents in advance), *Service Composer* in *Control tier* decided to invoke services, *callParent()* and *vibrate()*, so that their children' parents receive a call and the children's MIDs vibrate. In *Local Registry*, if there is more than one *callParent()* and *callTeacher()*, *Service Composer* determines and invokes the most appropriate service. An overall flow is represented in Figure 11. The figure only shows a case of invoking *callParent()*.

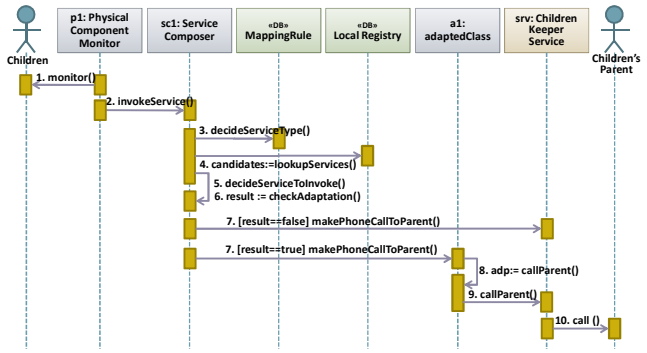

Figure 11. One Sequece Flow in *Safety Guard*

VI. CONCLUSION

Since the feature of *limited resource* forbids deploying complex computation on physical devices in CPS, many design issues are raised. To resolve these issues, we applied SOA or CC concepts. Compared to conventional CPS, we presented a three-tier architecture consisting of *Environment Tier*, *Control Tier*, and *Services Tier*.

By considering three tiers, we proposed key design methods. For the *Environment Tier*, we focused on allocating functionalities in an optimized way. For *Control Tier*, we dealt with monitoring physical components and services, supporting dynamic composition, ensuring service adaptability, and managing CPS in an autonomous way. And, for *Services Tier*, we presented a method to design reusable services. And to show applicability of our approach, we performed a case study. With our approach, we ensure that we can solve most of the key issues on CPS design and design CPS with high-quality more feasibly and practically.

ACKNOWLEDGMENT

This research was supported by the National IT Industry Promotion Agency (NIPA) under the program of Software Engineering Technologies Development.

REFERENCES

[1] Kim, J.E. and Mosse, D., "Generic Framework for Design, Modeling, and Simulation of Cyber Physical Systems," *ACM SIGBED Review*, Vol. 5, No. 1, Article No. 1, 2008.

[2] Prasad, V. and Son, S.H., "Design Suite for Deeply Embedded Cyber Physical Systems," *ACM SIGBED Review*, Vol. 5, No. 1, Article No. 2, 2008.

[3] Lee. E.A., "Cyber Physical Systems: Design Challenges," *In Proceedings of the 11th IEEE Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2008)*, pp. 363-369, 2008.

[4] Tan, Y., Goddard, S., and Perez, L.C., "A Prototype Architecture for Cyber-Physical Systems," *ACM SIGBED Review*, Vol. 5, No. 1, Article No. 26, 2008.

[5] Erl, T., *SOA Principles of Service Design*, Prentice Hall, 2007.

[6] Gillett, F.E., "Future View: New Tech Ecosystems of Cloud, Cloud Services, and Cloud Computing," *Forrester Research Paper*, 2008.

[7] Abdelwahed, S., Kandasamy, N., and Gokhale, A., "High Confidence Software for Cyber-Physical Systems," *In Proceedings of the 2007 Workshop on Automating Service Quality held at the International Conference on Automated Software Engineering (ASE 2007)*, pp. 1-3, 2007.

[8] Karsai, G. and Sztipanovits, J., "Model-Integrated Development of Cyber-Physical Systems," *In Proceedings of the 6th 6th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS 2008), Lecture Notes in Computer Science 5287*, pp. 46-54, 2008.

[9] West, R. and Parmar, G., "A Software Architecture for Next-Generation Cyber-Physical Systems," *Position Paper at the NSF Cyber-Physical Systems Workshop*, 2006.

[10] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns: Elements of Reusable Object-Orietned Software*, Addison Wesley, 1994.

[11] La, H.J. and Kim, S.D., "A Practical Framework of Realizing Actuators for Autonomous Fault Management in SOA," *In Proceedings of 2009 Congress on Services-I (SERVICES-I 2009)*, pp. 227-234, July 6-10, 2009.

[12] Kim, S.D. "Autonomous Service Management for Service-Oriented Architecture," *In Proceedings of the IEEE International Conference on Enterprise Systems and Applications (IEEE ICESA 2007)*, Dec. 2007, pp.5-12.

[13] Kim, S.D. and and Chang, S.H., "SOAR: An Extended Model-Based Reasoning for Diagnosing Faults in Service-Oriented Architecture," *In Proceedings of 2009 Congress on Services-I (SERVICES-I 2009)*, pp. 54-61, July 6-10, 2009.

[14] Her, J.S., La, H.J., and Kim, S.D., "A Formal Approach to Devising a Practical Method for Modeling Reusable Services," *In Proceedings of 2008 IEEE International Conference on e-Business Engineering (ICEBE 2008)*, pp. 221-228, 2008.

[15] Chang, S.H., La, H.J., and Kim, S.D., "A Comprehensive Approach to Service Adaptation ," *In Proceedings of IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2007)*, pp.191-198, 2007.

[16] Kim, S.D., Her, J.S., and Chang, S.H., "A Theoretical Foundation of Variability in Component-Based Development," *Information and Software Technology(IST)*, Vol. 47, p.663-673, July, 2005.

[17] Object Management Group, *CORBA Component Model Specification, version 4.0*, OMG, April 2006.