A New Framework for Configuration Management and Compliance Checking for Component-Based Software Development

Manali Chakraborty and Nabendu Chaki

Abstract Component-based software development (CBSD) decreases the time and cost for developing high quality software. However, with CBSD, the maintenance of the software is more difficult, as the whole system consists of several composite components. In this paper, a three-layer framework is proposed toward designing an efficiently configurable component-based system. We also developed an algorithm to identify the primitive and composite components that are related in terms of dependency. This helps managing multiple versions of a system. A smart meter system is considered as a case study. Our algorithm is executed on this component-based system using the semantic effect annotations of Business Process Modeling Notation (BPMN) to validate the results of our algorithm. The success reflects the effectiveness of the proposed algorithm toward identifying the components affected by a change in a simple way.

Keywords CBSD · Configuration management · Compliance · Version management

1 Introduction

Modern software systems become more large and complex, because of their improved performance, efficiency, and better quality. Also, the production costs and time for these systems should be minimized. Thus, the maintenance and modifications of those systems are also becoming more critical [1]. Traditional approaches for software development cannot deliver software in short deadlines and with lower costs. A new paradigm called CBSD is used to develop software with existing,

M. Chakraborty (✉) · N. Chaki
Department of Computer Science and Engineering, University of Calcutta, Kolkata, India
e-mail: manali4mkolkata@gmail.com

N. Chaki
e-mail: nabendu@ieee.org

R. Chaki et al. (eds.), *Advanced Computing and Systems for Security*,
Advances in Intelligent Systems and Computing 396,
DOI 10.1007/978-81-322-2653-6_12

already built and used components. In CBSD, the software systems can be developed by selecting off-the-shelf components from some component repository and then integrating them to build the intended software. The components can be developed by different developers using different languages and technologies [2]. Instead of building every software from scratch, CBSD reuses the components, modifies them to satisfy the requirements and then assembles them. This leads to lower cost, smaller development time and better quality of the software, as the components are already built and tested.

The differences between traditional software development (TSD) approaches and CBSDs are listed in Table 1. In CBSD, the management of different components and their versions is one of the most challenging tasks. To achieve this configuration management is used. Configuration management is the task of managing the configuration of different components in a system so that the system operates seamlessly. For a large and complex system, a systematic use of configuration

Table 1 Difference between TSD and CBSD

Property	TSD	CBSD
Development style	Each software is developed from scratch	Already existing components are assembled to build new software. Reusing of software components are the main theme of CBSD
Life cycle	In TSDS the different activities are, requirement analysis, feasibility study, design, coding, testing, maintenance etc.	Life cycle in CBSD consists of, finding components, selecting those that fit the requirements, adapting them, and replacing them with modified versions
Languages	Programming languages are used to implement the system	Primitive components are implemented using programming languages, and composite components are built using component description languages and architecture description languages
System construction	The system is usually implemented by a group of source code files which can be compiled and linked together to form the final system	System construction is a recursive process, in which, primitive components are used to construct composite components. Both primitive and composite components are used to construct larger composite components
Working team	There are engineering teams, which provide all the functionalities during the life cycle of software and end-users	There are component producer teams developing components; consumer teams developing software reusing components and hybrid teams that are both consumer, producer, and end-users

management is used to maintain the correct operability of the components. The different functions of configuration management are [3]: version management, change management, build management, release management, and workspace management.

Authors of paper [4–6] have discussed the various challenges of configuration management in CBSD. They also suggested that run time configuration is needed for CBSD and proposed a model for it. In [7], authors propose a component-based configuration management model, where the components are the integral logical constituents of the system. The model analyzes the relationship among the components and the configuration management part is dependent on that analysis. A model based on the component system and layered architecture is proposed in [8]. Authors claimed that this layered architecture improves reusability and maintainability of configuration management in CBSD. Another distributive, component-based layered model for configuration management in CBSD is proposed in [9]. The layered architecture makes this model easily adaptive, dynamic to changes and brings down the coupling of the system. In [10], dependency graphs identify different types of dependencies among components and analyze them. The graphs are used to facilitate maintenance by identifying differences, i.e., deviations of a configuration from a functioning reference configuration. Based on the unique features of CBSD, we summarize new requirements of CM for CBSD as follows:

(1) For component-based software development, the first step is to select a component from the existing component database. The owners of the database may update the components periodically. If there are more than one versions of a component between two baselines, then there will be two aspects for version management: either store the older versions in the repository, or replace the older versions by the new version. For the first case, the user can use older versions of a component if they want to. However, for the next situation, users are forced to accept the new versions of the components. In Fig. 1, two versions of component 1 exist between two baselines. If a user wants to use version 3 of component 1, then, it will allow doing so, if the older versions of the component are stored in the repository. Otherwise, it has no choice, but to work with the new version of component 1.

(2) Suppose two composite components cc1 and cc2 are dependent on primitive components pc1, pc3, pc5 and pc2, pc3, pc4, respectively, as in Fig. 2. Let us update primitive components pc2, pc3, pc4, and form a new base line for composite component cc2. Now, composite component cc1 is also dependent on primitive component pc3. So, for cc1 there exist two scenarios:

(i) If older versions of primitive components are replaced by the new versions of those components, then cc1 has to adopt itself with the new version of pc3. And the other primitive components of cc1, such as pc1 and pc5, may also need some up gradation to comply with the new version of pc3. Thus, the modification of one component can lead to modification of several components, which may or may not be directly linked with that component.

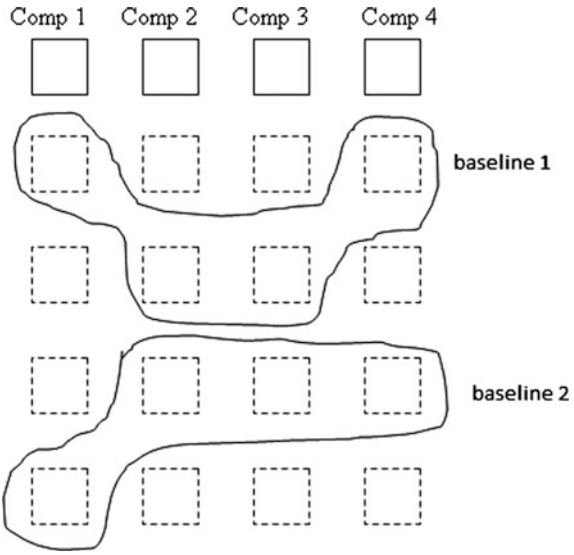

Fig. 1 Version management problem in CBSD

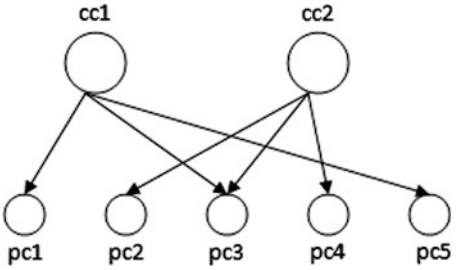

Fig. 2 Dependency between components

(ii) If older versions of primitive components are kept in a repository, then for a given time instant (for a baseline), the two composite components of a system will have two different versions of the same primitive components. This may lead to a compliance error.

In order to overcome these problems, we propose a new framework for providing configuration and compliance management of components of a system. This model consists of three layers: component management, configuration management, and compliance management. There exists some research work on configuration management in CBSD, but neither of them incorporates the idea of compliance management with that. Compliance is a very important factor in CBSD because the developer imports the components from outside. This makes them vulnerable for violating the business terms and policies of an organization. Thus, we propose a layered model which incorporates configuration management and compliance checking with CBSD. The component management layer deals with the selection of components and assembling them. It also modifies and replaces

components if necessary. The configuration management layer keeps tracks of the dependencies and relationships between the components. It also analyzes them for achieving maintainability. The compliance management layer is responsible for compliance checking of each component, as well as the total system. We also propose an algorithm to find the effect propagation within a system, when a primitive component is modified. The algorithm is able to identify sets of primitive and composite components, which may get affected with the modification of a certain primitive component. Thus, it helps to configure a system properly.

The smart metering system of smart grid architecture is considered as an application domain for this proposed framework. First, we identify some basic functionalities of smart metering system and consider them as composite components. To achieve each of these functionalities, several small and atomic processes are needed to be executed. We consider all those atomic processes as primitive components. All the composite components, with their respective primitive components are stored in a tabular data structure. When a change request is placed for a primitive component, then our algorithm is used to find how the effects of this change propagate through the system. As an outcome of the algorithm, we can identify the sets of primitive and composite components, which may get affected by the proposed change.

In order to validate the proposed algorithm, Business Process Modeling Notation (BPMN) has been used. BPMN is an agreement between multiple modeling tools' vendors, who had their own notations. BPMN uses a single notation that is understood by all the end-users. It can be used to analyze, simulate, or execute a particular business model. A business process model describes the ordered sequence of different tasks within a process and how the process achieves its objectives [11].

BPMN is an internationally accepted, model-independent tool, which can create a bridge to reduce the gap between business processes and their implementations by providing a unified and standardized graphical representation of any business model. Thus, as a basis of our proposed framework, we use BPMN, to graphically describe smart metering systems, and analyze the effect of component changes on the whole system. When a change request is placed, then the aftereffects of the change for each individual component, as well as the cumulative effect on the whole system is analyzed using semantic effect annotations [12]. This helps identifying the affected components through change propagation and how the system can be reconfigured for a particular change request. The objective is to compare the results produced by the proposed algorithm vis-à-vis finding from BPMN.

An obvious question may arise in this situation: if we can achieve the goal of our algorithm using the semantic effect annotations in BPMN, then why a new algorithm is required at all? Identifying the effects of a change by semantic effect annotation in BPMN needs a certain amount of knowledge about both BPMN and propositional logic. Besides developing the BPMN model for a process and maintaining the semantic effect annotations is a complex task as changing any primitive component results in changing its immediate effect. This change in immediate effect needs to be propagated throughout the system so that the cumulative effects evaluated at

different points within the system remain consistent with this change. Also, using BPMN requires maintenance of a graphical representation of the system. On the other hand, our algorithm is inherently simple as it does not require any graphical representations of the system or semantic effect annotations. A simple tabular data structure is sufficient for the execution of the algorithm. As a result, it is a less complex and more preferable solution compared to BPMN.

The rest of the paper is organized as follows: Sect. 2 describes the framework, Sect. 3 explains the functionalities of smart metering system, and the effect of our framework with the help of BPMN and Semantic effect annotation process. We discuss on the future expansion of the work and draw conclusions in Sect. 4.

2 Working Principle of Proposed Framework

In this paper, we proposed a new framework for configuration of components of a component-based software with run time compliance checking. The proposed model has three different parts: (1) component management, (2) configuration management, and (3) compliance management.

2.1 Component Management

The component management part basically deals with the selection of composite and primitive components and maintains their relationships in the form of a list. The functions of the component management are:

Select components	Modify components	Integrate components	Replace components

Select Components The component manager first identifies the composite components of the system. Then for each composite component, primitive components are selected from the component repository.

Modify Components It is not always possible to find the exact component, which meets the requirements of the system. So, then the component manager modifies the components according to the requirements and adapts them to the system.

Integrate Component After collecting all the primitive components, the component manager integrates those primitive components to develop a composite component. The interconnections and dependencies between the composite components are also maintained by the component manager.

Replace Component The component manager also replaces the older versions of a component by the newer and upgraded versions of that component.

The component management layer maintains a data structure for storing the composite components and the primitive components used for each composite

component. Let, there be n composite components, C1 to Cn. For every Ci ($1 \leq i \leq n$), component manager maintains a list of all of its primitive components.

Structure *Component_Relation* C
{

Primitive Component P_1;
Primitive Component P_2;
.
.
Primitive Component P_n;

}

2.2 Configuration Management

The configuration management part deals with the version management of each primitive components and how it affects the whole system. Since primitive components are interrelated, modification in one primitive component leads to the modification of its dependent components. The functions of configuration management are:

Monitor	Select a component for modification	Identify all the related components	Modify	Report to component management	Store

Monitor the configuration manager monitors the whole system to assure that its working properly and consistently.

Select a component for modification While monitoring the system, the configuration manager also maintains a database for storing the versions of each component. If a new version of a component arrives in the market, then the configuration manager identifies that component for modification.

Identify all the related components Modifying one primitive component at run time may affect all the other primitive components related with that component, and the composite components which are associated with them. So to maintain consistency it is necessary to modify all the other components. Configuration manager uses an algorithm to identify the related components of an primitive component.

Modify After identifying the components, the configuration manager modifies the components accordingly.

Report to Component management Then configuration manager reports to the component manager about these modifications. The component manager then

checks the newly modified components and sends them to compliance manager to make sure that they comply with the business rules of the company.

Store after the compliance checking of the modified components, the configuration manager stores the new versions of those components in a database.

Suppose a primitive component P_j has been modified and a new version of P_j, i.e., $P_j.1$ is introduced. The purpose of this algorithm is to identify the related primitive as well as composite components.

Algorithm: ***Dependency_Analysis*****(C, P_j, N)**
Input: C: Structure *Component_Relation* C.
P_j: Primitive component which has been modified.
N: total number of Composite Components.
Output: P_{Array} - Array for storing primitive components, which are related to a particular primitive component Pi.
C_{Array} - Array for storing the Composite Components, which are related with a particular Primitive Component Pi.

```
{
   for i: = 1 to N
   {
      traverse Ci;
      if (Pj is in the list of Ci and Ci is not in CArray)
      {
         CArray: = Ci;
         for (the rest of primitive components in the
         list of Ci)
            if (they already exist in the PArray)
               break;
            else
               put the primate component in PArray;
      }
   }
      end;
}
```

Let us assume that a system has eight composite components. Figure 3 describes the structure for eight composite components. Suppose primitive component P5 has been modified due to some reasons. Therefore, a new version of P5 is introduced as P5.1. In order to maintain the concurrency and compatibility, we must check the other primitive components that are related to P5. In cascade, the composite components which depends on those primitive components will also be checked.

First, we find P5 from the component table. It has been found in the list of C1. Then C1 is added in the C_{Array}, and all other primitive components of C1, i.e., P1 and P2 are added in the P_{Array}. Next, P5 is also in the list of C3. So we put C3 in C_{Array} and P6 in P_{Array}. P5 is not connected with any other composite component. So we take the second element from the P_{Array}, i.e., P1, and repeat the same procedure. P1 is not connected with any other composite component, so we move on to the next primitive component in P_{Array}, P2. P2 is in the list of C4 and C7. So we put both of them in C_{Array}, and add their primitive components, i.e., P8 and P4 in P_{Array}. The next primitive components in P_{Array} are P6, P8, and P4. Since they are not in the list of any other composite components, the procedure is terminated. Figure 4 shows the content of P_{Array} and C_{Array}.

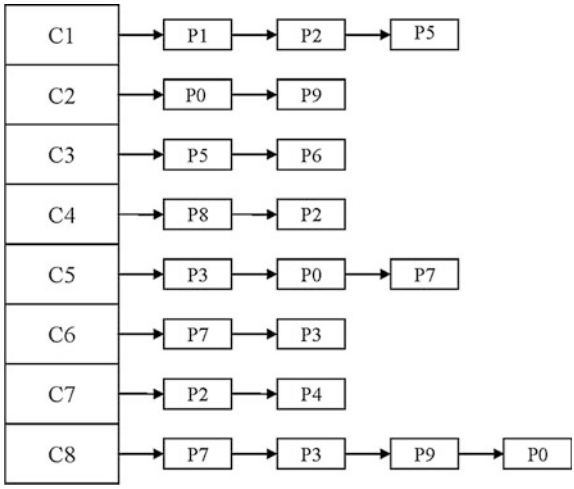

Fig. 3 Component relation structure

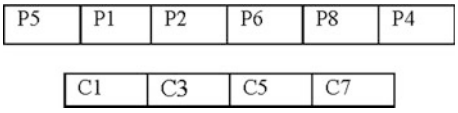

Fig. 4 Contents of P_{Array} and C_{Array}

2.3 Compliance Management

The compliance management layer is responsible for checking the compliance of each individual component and as well as the whole system. It also records new compliance rules through the development process of the system. The functions of the compliance management part are:

Check primitive components for compliance	Record new compliance rules	Monitor the integrated system for compliance

Check primitive components for compliance When component manager imports the primitive components from outside, then the compliance manager checks every primitive component for compliance. If they do not comply, then the compliance manager reports to the component management layer, and the component management layer modifies that component accordingly, so that it can comply with the system.

Record new rules While integrating the primitive components, it is sometimes necessary that the components, both primitive and composite, should comply with

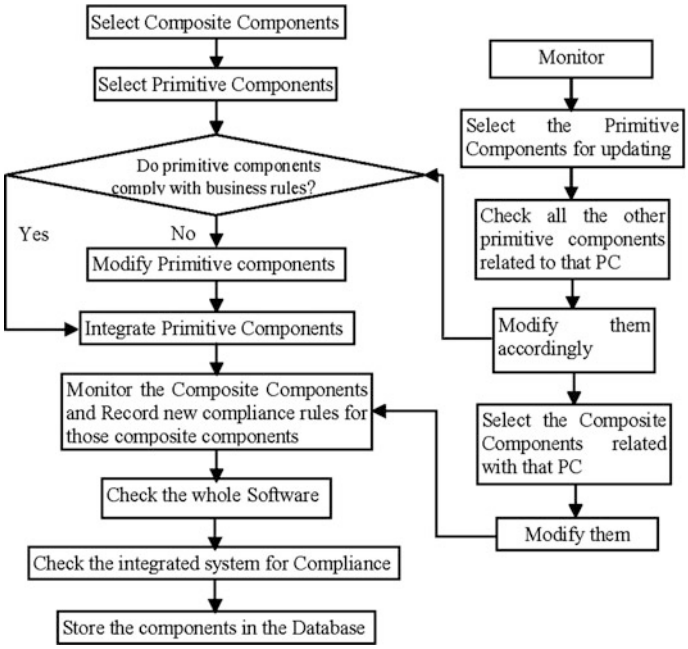

Fig. 5 Workflow diagram of proposed framework

some new rules for successful execution. Thus, another function of compliance manager is to record new compliance rules at run time and keep the business rule database up to date with each change in the system.

Monitor the integrated system for compliance Compliance of each primitive and composite component does not always imply that the whole system is also compliant with the business rules. Therefore, after checking the primitive and composite components for compliance, the compliance manager monitors the whole integrated system for compliance. A baseline is approved only when the system is compliant with the business policies.

The workflow model of the proposed framework is depicted in Fig. 5.

3 Application of this Model in Smart Grid Architecture

In this paper, we consider the smart meter communication architecture of smart grids as an application domain of our proposed model. A smart grid is an intelligent electricity network that integrates the actions of all users connected to it and makes use of advanced information, control, and communication technologies to save energy, reduce cost and increase reliability, and transparency [13].

Smart meter is an advanced energy meter that measures the energy consumption of a consumer and provides added information to the utility company as compared to a regular energy meter [14]. The smart meter communication architecture typically consists of four different components: smart meter, smart energy utility network, DCU (data collection unit), and MDMS (Meter data management System) [15].

The entire scenario is modeled using BPMN. When a change request is placed, then the after effects of the change for each individual component, as well as the cumulative effect on the whole system is analyzed using semantic effect annotations of BPMN. This helps identifying the components affected by the change and how the system can be reconfigured for a particular change request.

We assume that, there are five basic services provided by a smart meter. We consider these five services as five composite components and each composite component further decomposed into several primitive components.

Table 2 provides a detail list of the entire composite and their primitive components for a smart metering system. Now we apply the proposed algorithm on this system and analyze the effect of changing a primate component on the system.

If, primitive component P5 is modified, then the contents of P_{Array} and C_{Array} will be,

$$P_{Array} = P6, P7, P8, P1, P9, P15, P16, P2, P3, P4, P10, P11, P12, P13, P14.$$
$$C_{Array} = C2, C3, C5, C1, C4.$$

Thus, if P5 is modified, then we have to check all the primitive and composite components to check who also need modification. Figure 4 shows the BPMN diagram of the smart meter system. BPMN provides a graphical diagram of how different objectives can be achieved in a business process, with enough information, so that the process can be analyzed, simulated and executed. There are different elements in BPMN—activities, events, gateways, and connectors. A connector links activities, events and gateways and shows the control flow relation. An event can be a start event (start of the process), end event (end of the process), or an intermediate event, that can either be some messages or a timer or error. An activity or a task is an atomic activity and stands for work to be performed within a process. Gateways determine the branching, forking, merging, and joining of paths [11, 16] (Fig. 6).

Immediate effects can be described as the outcome of execution of an activity. This model requires the designers to provide the immediate effects of each activity. Then, the cumulative effect of each component can be calculated by accumulating the immediate effects [12, 17].

In Fig. 4,

- e1 to e16 are the immediate effect of primitive components P1 to P16, respectively.
- CEC1 to CEC5 are the cumulative effect of composite components C1 to C5, respectively. The arrows toward CEC1 to CEC5 mark the points where the cumulative effects have been calculated.

Table 2 Component structure of smart metering system

Composite components	Primitive components
C1: Generate the total electricity consumption of a user	P1: Decode Receive message from DCU
	P2: Collect the total unit of usage
	P3: Generate the bill
	P4: Send message to DCU
C2: Send SMS, if the consumption unit of a user exceeds its previous bill	P5: check the current unit of usage, with previous bill
	P6: Generate an alert message for excess bill amount
	P7: Generate a intermediate bill
	P8: Send message to the user
C3: Alert user before power cuts	P1: Decode Receive message from DCU
	P8: Send message to the user
	P9: Generate an alert SMS for power cut
C4: Services provided for users, who generate electricity in their own houses	P10: Check the electricity generation of a home
	P11: Draw current from home electricity source
	P12: Draw current from outside electricity source
	P13: Check if, generated electricity is sufficient for the home
	P14: calculate the amount of surplus energy and generate a message
	P4: Send message to DCU
C5: take necessary actions, if DCU reports a power shortage	P1: Decode Receive message from DCU
	P8: Send message to the user
	P15: Generate an alert SMS for power shortage
	P16: cut off electricity to some appliances after certain time period

$$\text{Cumulative effect of C1 (CEC1)} = (e1 \wedge e2 \wedge e3 \wedge e4)$$
$$\text{Cumulative effect of C2 (CEC2)} = (e5 \wedge e6 \wedge e8) \vee (e5 \wedge e7 \wedge e8)$$
$$\text{Cumulative effect of C3 (CEC3)} = (e1 \wedge e9 \wedge e8)$$
$$\text{Cumulative effect of C4 (CEC4)} = (e10 \wedge e12) \vee (e10 \wedge e11 \wedge e13 \wedge e14 \wedge e4) \vee (e10 \wedge e11 \wedge e13 \wedge e12)$$
$$\text{Cumulative effect of C5 (CEC5)} = (e1 \wedge e15 \wedge e16 \wedge e8) \vee (e1 \wedge e15 \wedge e8 \wedge e16)$$

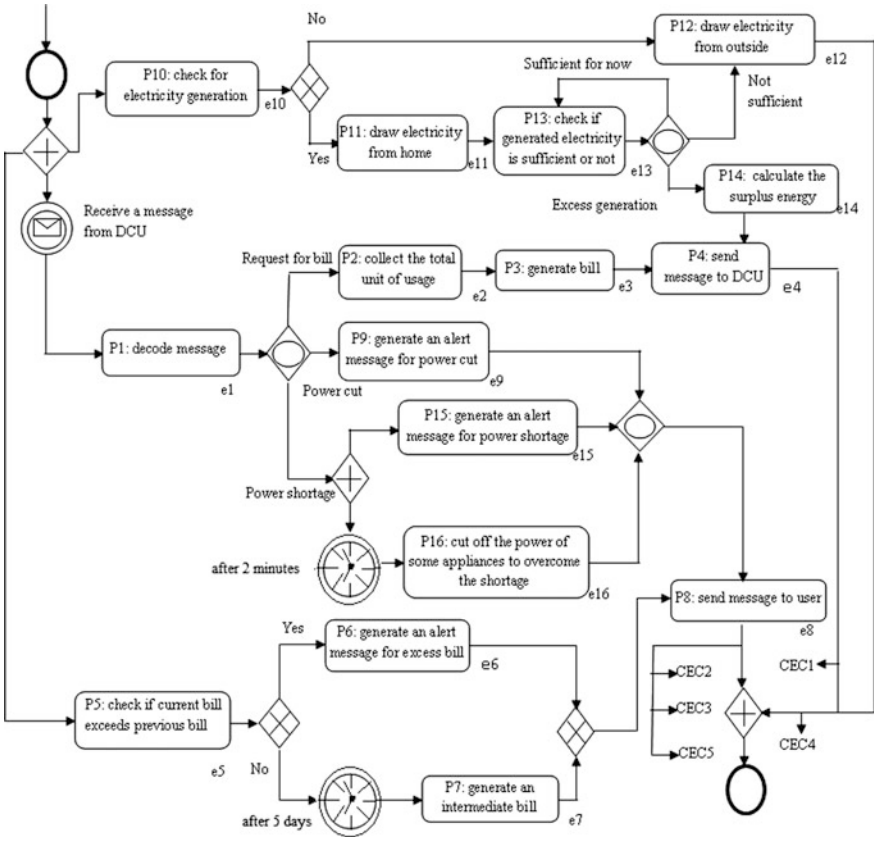

Fig. 6 BPMN diagram of a smart meter

Again, if a change request for P5 is made, then from the diagram and the cumulative effects, we can conclude that,

- Cumulative effect of C2 may get affected, as well as the immediate effect of other primitive components of C2, i.e., e6, e7, e8, and the immediate effects of P6, P7, and P8, respectively.
- Now, P8 is further used in C3 and C5. Hence, if the immediate effect of P8 changes, due to P5, then it may also affect the immediate effects of P1, P9, P15, and P16.
- Again P1 also had contributions in the cumulative effect of C1. Thus, P2, P3, P4 might be affected.
- P4 is also used in C4. So, P10, P11, P12, P13, P14 might also be affected.

Hence, we may conclude that, the BPMN with semantic effect annotation confirms the result of our algorithm.

4 Future Work and Conclusions

One of the most important criteria for a CBSD is to comply with the business policies, rules and regulations of a company. Compliance often refers to the validation of a system against some legal policies, internal policies, or some basic design facts [6]. Compliance checking can be of two types: compliance by detection and compliance by design. In Compliance by detection method, the existing system is checked thoroughly to detect whether it violates any rules or not. If it does not comply, then corrective measures are taken to make it compliant. In compliance by design method, the system is developed, by taking into account the business rules. Thus, the system is designed in such a way, that it can comply with the rules [18]. In CBSD, the system is not developed from scratch. Thus, the compliance by design method does not suit CBSD. Hence, in CBSD, compliance by detection method is used. As an extension of this framework, we would like to work on the detailed working principle of the compliance layer.

In this paper, a new framework for configuration of components with compliance checking is proposed. This framework considers two main problems of CBSD: maintenance and compliance, and solves them by incorporating both configuration management and compliance management with CBSD. In this paper, we consider two-level hierarchy between components, i.e., all the composite components are developed using primitive components. However, the level of hierarchy can easily increased in this model, so that we can consider a scenario where composite components are again assembled together to develop another composite component.

In Sect. 2, requirements for configuration management for CBSD is discussed. Two main problems have been highlighted. One is due to version management, and another is due to the complex and nested relationship between the primitive and composite components. The model in this paper is able to overcome these problems. In this model, the component management layer replaces each component with its latest version once it is accepted by the compliance layer, but the configuration management layer stores all the versions of a component to a database. Thus, the active system is always executed with current versions of each component, but the older versions are also stored in the database.

Also, the component management layer uses a tabular data structure and the configuration management layer uses an efficient algorithm to search all the related primitive and composite components for a particular primitive component. This helps the model to perform efficiently and provides an easily maintainable and compliant system.

Although it is not a theoretical proof for correctness of the proposed algorithm, the validation using BPMN indeed shows the effectiveness of the new algorithm. The proposed methodology builds the foundation for several meaningful extensions in future. We want to apply this model to the entire smart grid architecture as a future work.

5 Acknowledgment

This work is a part of the Ph.D. work of Manali Chakraborty, a Senior Research Fellow of Council of Scientific and Industrial Research (CSIR), Government of India. We would like to acknowledge CSIR, for providing the support required for carrying out the research work.

References

1. Crnkovic, I.: Component-based software engineering—new challenges in software development. J. Comput. Inf. Technol. CIT 11. **3**, 151–161 (2003)
2. Pour, G.: Component-based software development approach: new opportunities and challenges. In: Proceedings Technology of Object-Oriented Languages. TOOLS 26. pp. 375–383 (1998)
3. Estublier, J.: Software configuration management: a roadmap. In: Proceedings of 22nd International Conference on Software Engineering, the Future of Software Engineering. ACM Press, New York (2000)
4. Larsson, M., Crnkovic, I.: Development experiences of a component-based system. In: 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems ECBS (2000)
5. Larsson, M., Crnkovic, I.: Component configuration management. In Proceedings of ECOOP Conference, Workshop on Component Oriented Programming Nice, France (2000)
6. Lohmann, N.: Compliance by design for artifact-centric business processes. In: 9th International Conference on Business Process Management, pp. 99–115 (2011)
7. Hong, M., Lu, Z., Fuqing, Y.: A component-based software configuration model and its supporting system. J. Comput. Sci. Technol. **17**(4), 432–441 (2002)
8. Mao., M, Jiang, Y.: A new component-based configuration management 3C model and its realization. In: ISISE, International Symposium on Information Science and Engineering, vol. 1, pp. 258–262 (2008)
9. Ruan, L., Yong, Z.: A new configuration management model for software based on distributed components and layered architecture. Parallel Distrib. Comput. Appl. Technol. 665–669 (2003)
10. Larsson, M.: Applying configuration management techniques to component-based systems. Licentiate Thesis Dissertation, Department of Information Technology Uppsala University, vol. 7 (2000)
11. Object Management Group: Business Process Modeling Notation (BPMN) Version 1.0. OMG Final Adopted Specification. Object Management Group (2006)
12. Hinge, K., Ghose, A., Koliadis, G.: Process SEER: a tool for semantic effect annotation of business process models. In: Thirteenth IEEE International Enterprise Distributed Object Computing Conference (EDOC) Los Alamitos, USA, pp. 54–63. IEEE (2009)
13. White Paper by United States Agency for International Development, USAID India: The smart grid vision for India's power sector (2010)
14. Kim, M.: A survey on guaranteeing availability in smart grid communications. Adv. Commun. Technol. (ICACT) 314–317 (2012)
15. Jung, N.J., Yang, K., Park, S.W., Lee, S.Y.: A design of ami protocols for two way communication in K-AMI. In: 11th International Conference on Control, Automation and Systems, pp. 1011–1016 (2011)
16. Goel, N., Shyamasundar, R.K.: An executional framework for BPMN using Orc. APSCC, pp. 29–36. IEEE (2011)

17. Koliadis, G., Vranesevic, A., Bhuiyan, M., Krishna, A., Ghose, A.: Combining i* and BPMN for business process model lifecycle management. In: BPM'06 Proceedings of the 2006 international conference on Business Process Management Workshops, pp. 416–427 (2006)
18. Sackmann, S., Kahmer, M., Gilliot, M., Lowis, L.: A classification model for automating compliance, pp. 79–86. CEC/EEE. IEEE (2008)