Towards Knowledge-intensive Software Engineering Framework for Self-Adaptive Software

Hyo-Cheol Lee
Dept. of Computer Engineering, Ajou University
NiSE Research Group
Suwon, South Korea
mytion7@ajou.ac.kr

Seok-Won Lee
Dept. of Computer Engineering, Ajou University
NiSE Research Group
Suwon, South Korea
leesw@ajou.ac.kr

***Abstract*—A self-adaptive system reacts to the changing environment by modifying its functionality in relation to the encountered state of the environment. In order to adapt to a new situation, such system goes through many decision points during the adaptation process. Knowledge forms the basis of decision making within the adaptation process. There are already many existing self-adaptive system frameworks. However, these frameworks have limitation in the way they represent the rationale for adaptation and the semantics behind the knowledge they use. This paper takes a step forward by proposing a knowledge-intensive adaptation framework to both manage knowledge and support the analytical decision making process. The proposed approach represents the adaptation knowledge by using ontology which helps to organize, analyze and extend knowledge. Ontology is able to represent the semantics behind knowledge and provide the evidence for the adaptation. The proposed approach uses a special ontology named the Adaptation Problem Domain Ontology. It specifies the system goals, features, architectures, and the relationship between them. This ontology is used to answer the problem of adaptation at each decision point and determine the appropriate system structure by reasoning the semantics behind knowledge. Thus, the system can consider the semantics behind knowledge for adaptation, and then the stakeholders can understand the adaptation process. We apply the proposed framework to the smart grid domain and show how the system adapts to a new situation using rationale for adaptation and the semantics behind the knowledge.**

***Index Terms*—Self-adaptive system, decision making, ontology, goal model, feature model, role-based architecture**

I. INTRODUCTION

As humans interact with changing environments, so a system encounters many different situations which demand different requirements or capabilities. Thus, a system should be able to provide a specific functionality, which is appropriate to the encountered situation, for the user. This has been a great motivator for the development of self-adaptive systems. A self-adaptive system can handle many situations by modifying the system goals, architecture, and functionality in response to changing environments without any human intervention [1]. For self-adaptation, the MAPE-K (Monitor/Analysis/Plan/ Execute and Knowledge) process is widely used [2]. Following this process, the system encounters many decision points that determine what is appropriate in a given situation and achieves the emergent system objectives [3]. At that time, knowledge plays a critical role as the foundation for decision making [4]. Many kinds of knowledge can be used to determine the results and quality of the entire self-adaptation process. That is, the way to use and represent knowledge is important in the self-adaptation.

DOI reference number: 10.18293/SEKE2015-222

Many existing self-adaptive system frameworks already regard knowledge as the basis for the adaptation. However, in these frameworks, knowledge is considered as predefined rules, logic and formulas mapping between input and system structures and functionality [5][6][7]. The adaptation process is therefore simplified as mapping between problem and predefined solution. This is similar to the black box testing, where the tester does not consider the internals of the system during testing. This results in the semantics and rationale behind the adaptation to be ignored and implicitly implied. The rationale aspect behind adaptation is essential for stakeholders to understand the reason behind decision making. Existing frameworks are insufficient to illustrate the semantics and rationale behind the adaptation process.

In this paper, we propose the NiSE (kNowledge-intensive Software Engineering) framework for self-adaptive system. The proposed framework adopts an ontological approach to represent knowledge for the adaptation process. Various types of knowledge needed for self-adaptation are systematically organized, connected, and used in the form of ontology. So, using this ontological approach, we are able to provide knowledge-intensive adaptation process including the decision making process which uses the rationale and semantics behind the adaptation [8]. In this adaptation process, the decisions do not just follow predefined logic or formulas as seen in existing approaches [10][11][12], but infer the appropriate ones using the relationship among knowledge. For that, the APDO (Adaptation Problem Domain Ontology) is a key component. APDO is special ontology containing adaptation knowledge such as system goals, features, architecture, and their relationships. During the adaptation, a system infers the appropriate decision using APDO by answering a question at each decision point. It gives support to know which knowledge has been used in the adaptation process. Thus, the proposed approach supports a comprehensive adaptation process through

an ontological approach, and helps stakeholders to understand the rationale and semantics behind the adaptation [9].

This paper is organized as follows: Section 2 introduces the application domain, which is used to illustrate the proposed approach. In Section 3, the proposed approach is described with the help of a case study. We examine other frameworks in Section 4. Section 5 concludes with future works.

II. Application Domain

In order to verify the applicability of the proposed NiSE framework, we have used a case study in the smart grid domain. A smart grid is next generation electricity grid which simultaneously interacts with demand and supply side using their information [24]. The behavior of a smart grid corresponds with that of a self-adaptive system. The smart grid system also includes and manages many kinds of knowledge such as domain, context, and system structure for adaptation. This provides a domain that is suitable for us to test the feasibility of the NiSE framework.

In this case study, APDO for the smart grid includes the following knowledge: 1) goal model for what a smart grid wants to achieve, 2) feature model to represent variability of a smart grid behavior and component, 3) role-based architecture model which a smart grid can have, and 4) other context and policy related to the smart grid domain. These are correlated with each others and used to make an appropriate decision.

We will use the electricity shortage scenario in this case study [25]. In the smart grid, backup power is stored for emergency situations and should be maintained with certain proportions. Based on the amount of a backup power in a smart grid, there are three states of power warning: *Ready*, *Warning*, and *Severe*. *Ready* is safe state with enough backup power and it maintains its goal and policy. *Warning* is careful state where it needs volunteers to reduce electricity consumption. *Severe* is the most critical state and it is compulsory to regulate electricity consumption. In each state, there is a certain policy to return to the *Ready* state. Thus, maintaining *Ready* state is one of the goals of a smart grid. It means that if the backup power is decreased and the power warning state is changed from *Ready* to *Warning* or *Severe*, a smart grid should change its behavior based on a policy in order to adapt to new situation.

In the case study, we assume that the energy consumptions on the end-users side is suddenly increased due to unexpected weather change. It causes the usage of a backup power to resolve the emergent situation and changes power warning state from *Ready* to *Warning*. A smart grid system monitors these changes and reconfigures its goal, feature, or architecture for return to *Ready* state without any failure. In the next section, the details of NiSE framework is described and explained based on this case study.

III. NiSE Framework For Self-Adaptive System

NiSE framework for self-adaptive system is proposed to deal with knowledge aspect in the adaptation process and improve the stakeholders' understanding of adaptation by considering the rationale behind the adaptation. The NiSE framework mainly focuses on two perspectives: 1) adaptation knowledge and 2) associated adaptation process using that knowledge.

A. Adaptation Knowledge Perspectives

In the perspective of adaptation knowledge, we introduce APDO for the knowledge base of the self-adaptation. APDO is a special ontology, which defines knowledge including system structure, rules, and relationships between them [22]. Figure 1 describes the relationships among the various kinds of knowledge. It shows not only the main system structures such as goal, feature, and role architecture, but policy, software engineering process, context, and domain knowledge as well. These kinds of knowledge are used to determine the system behavior and the system architecture. By using these multi-dimensional relationships among many different types of knowledge for the adaptation, we are able to understand the internal process of decision making for the adaptation and support for the stakeholders to comprehensively understand the rationale behind the adaptation.

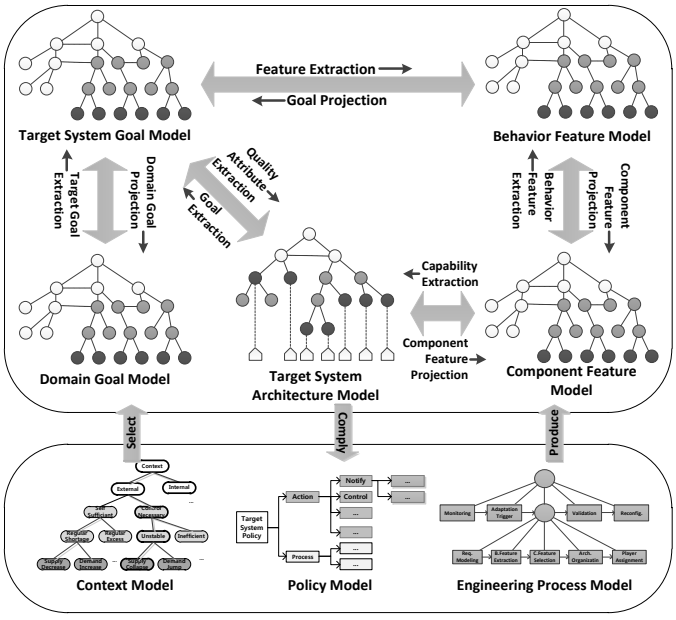

Figure 1 Overview of Adaptation Problem Domain Ontology

Among many types of knowledge, goal, feature and architecture models are directly related to the system structure. As we move from goal models to architecture model, the degree of abstraction is decreased and the details of the system structure are extracted. In order for each model to be associated, we explain the meaning and characteristics of each model.

Goal is the objective that the system wants to achieve [5]. It is used to represent the system's functional and quality requirements [14][15]. In NiSE framework, there are two types of goal model: domain goal model and target system goal model. Domain goal model has all possible goals that a system can achieve. As a subset of domain goal model, target system goal model only includes the goals that the system needs to achieve in given situation.

Goal model is the highest level of abstraction in NiSE framework. When a system goal changes, the purpose of a

system behavior also changes. If a system needs to change its goal, it should find a new goal, which can resolve the problem in a new situation, among domain goal model. Thus, setting domain goal model is defining the available adaptation strategies that a system can have.

Feature is defined as "*A prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems*" [16] and used to represent the system variability and commonality [17]. In NiSE framework, feature model is used not only to represent the variable points at which the system can have diverse options of its functionalities or architectures, but also to reduce the abstraction gap between goal model and architecture model.

For this purpose, NiSE framework includes two types of feature model: behavior and component feature model. Behavior feature model represents atomic actions and component feature model represents functional modules that realize those atomic actions. Behavior feature model is close to goal level and component feature model is close to architecture level. Using these models, a system can connect goal problem space and architecture solution space smoothly and represent variability and commonality with specific articulation [7].

The NiSE framework includes the system architecture using the role-based design approach [18]. It has many advantages to specify adaptive architectural design. Role is the abstract architecture unit, which does not exist in real world. The system is composed with the organization, which is comprised of several roles. The real system components play certain role to make a complete organization. This mapping is separately processed with constructing organization. Therefore, late binding between role and player is possible. It makes loose coupling between the system architecture and real implementation, and flexible architecture to easily change the system components [19][20].

Role model can represent quality requirements of the system through a contract. A contract is the specification of the interaction between roles [21]. A contract includes the process and the measurement. A process describes how the roles interact with each other and the measurement specifies achieving a contract. By measuring the degree of satisfaction of a contract, we can quantify the quality requirements as well.

Furthermore, knowledge of context, policy, software engineering and etc. are able to be represented following diverse models and standards. The form of these kinds of knowledge is not strictly restricted. Furthermore, in APDO, the system engineer can define and add new knowledge.

When a system encounters decision point, including a set of decision questions, the system queries APDO using the above mentioned knowledge as a form of ontology for determining appropriate decision. For example, in order to answer the question "Whether the current structure is in need of adaptation?", first of all, the system checks whether current system structure is appropriate for a given context or not. For that, the relations between context model and the system structure models are used to infer the answer. If the situation is changed, the system determines the level of adaptation based on the different system structure models and starts adaptation to satisfy new objective of the current situation by changing its goal, feature, role-based architecture or all of them.

The advantages of APDO are 1) supporting intuitive way to manage adaptation knowledge and 2) providing the evidence of the decision making during the adaptation process. When knowledge is extended and modified, the engineer has a trouble to predict the available situation and architecture based on new knowledge. Thus, it takes a lot of time and effort to infer the available situations and appropriate architecture corresponding to each situation [23]. However, if the engineer uses ontology, the engineer just defines knowledge and relationship among them in ontology. And then, the unpredictable and emergent solutions, which were difficult to determine by human, can be automatically inferred by the system. Besides, because many kinds of knowledge are used, it is able to provide the evidence of the adaptation to understand the rationale and semantics of the adaptation. This enhances the traceability between the situation and the adaptation outcome. It makes the stakeholders understand the adaptation process and application result.

The system engineer or domain experts define APDO, because it needs many kinds of knowledge of the system and domain. The system structures such as goals, features and architecture models have formalized engineering method which helps to define them. All models are converted to ontology classes using those meta-models and the relations between them are represented as object and data properties in ontology. Other knowledge such as policy and context are also illustrated in ontology based on the engineer-defined models. Therefore, the preprocessing of knowledge is required.

B. Adaptation Process Perspectives

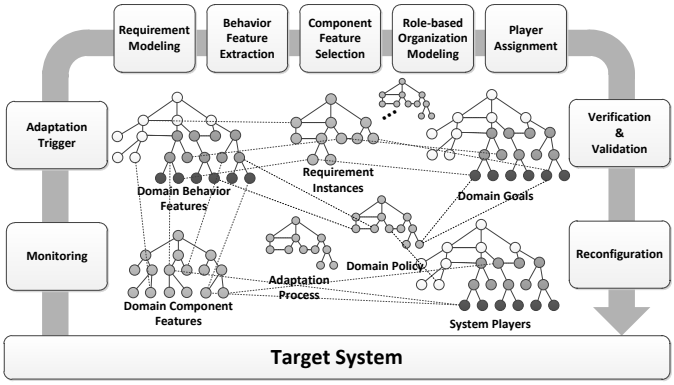

Figure 2 NiSE Adaptation Process

With the purpose of making a proper decision based on knowledge, Figure 2 shows the NiSE adaptation process. The adaptation starts from monitoring the environmental factors to reconfiguring current system architecture into the inferred system architecture. By following this process and answering the adaptation questions using knowledge, a system can make an appropriate decision, and then consequently adapt to the new situation. Each adaptation phase has unique decision points and several adaptation questions for making a decision. For instance, in the scenario described in Section 2, when the weather suddenly changes, the system can raise a question such as "Whether it is needed to adapt?" And then, through

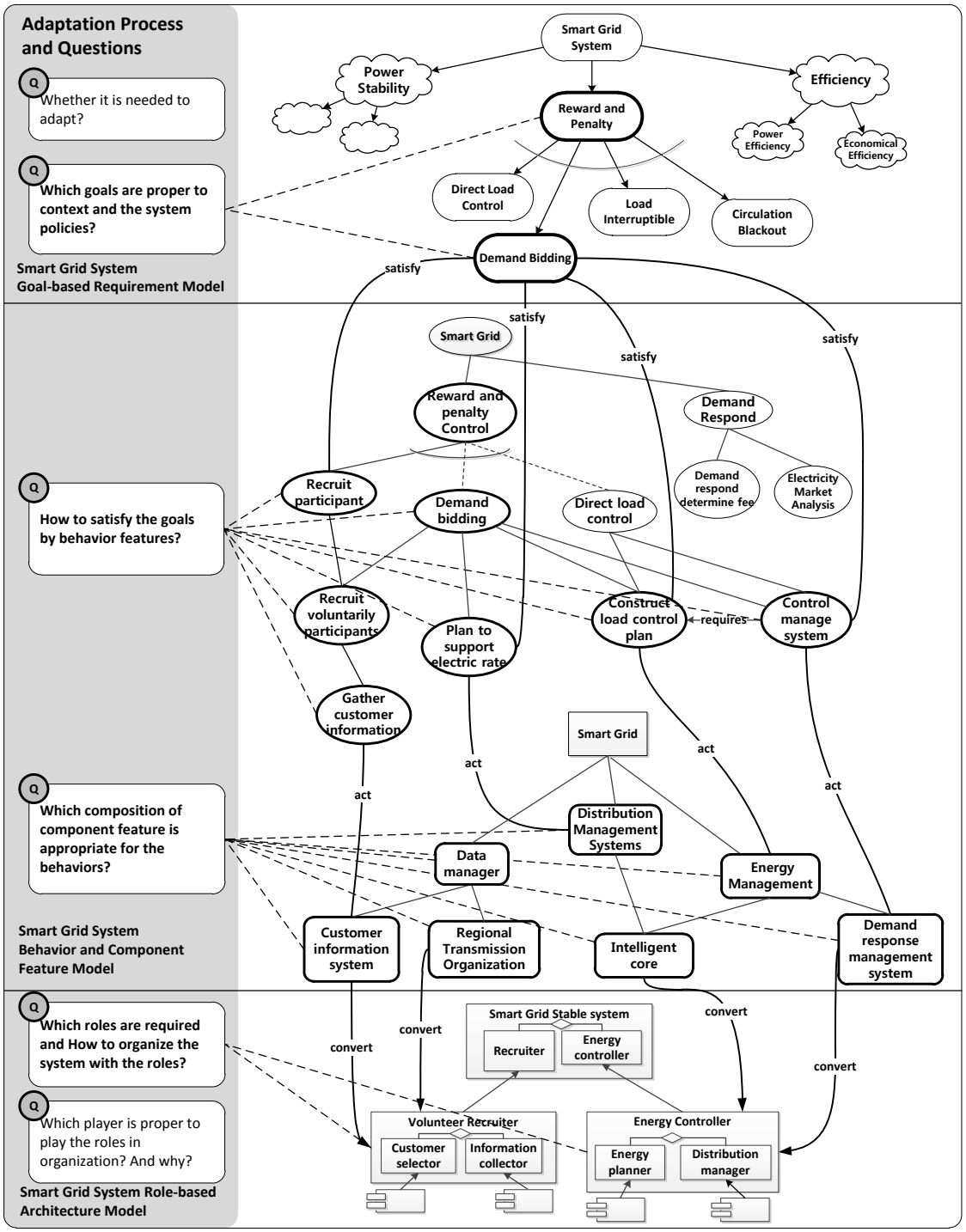

Figure 3 Example of NiSE Adaptation Process Using APDO

knowledge of context, policy and goals and the relationship between them in APDO, the system answers these questions and makes an appropriate decision. These decisions finally affect the system structure in order to satisfy new policies, contexts or requirements.

In order to understand the knowledge-intensive adaptation process, we show a simple example of the adaptation process using APDO. In this example, we define APDO with 70 classes, 68 object properties, and 25 data properties with respect to the smart grid system's context, policy, goal, feature and architecture to answer the questions shown in Figure 3 in accordance with the scenario in Section 2.

Figure 3 shows the adaptation process with questions at each decision point. Main adaptation process including from

goal-oriented requirement modeling to role-based architecture design is shown based on predescribed scenario. The blocks in Figure 3 represent goals, behavior features, component features, and organizations with roles. Each goal is satisfied by behavior features. These behavior features are also performed by component features. Based on selected goals, behaviors and components, which are able to perform the given behaviors, are determined. Lastly, these features are connected to organization and role which are composed of the corresponding capabilities.

In the scenario, energy warning state is changed from *Ready* to *Warning*. To address this change and return to a stable state, the system should increase backup power and decrease current usage of electricity. This is a smart grid domain policy used when energy warning state is changed to *Warning*. For this scenario, APDO includes several knowledge areas such as knowledge of policy, context, and system structure with three abstraction levels (Goal, Feature, and Role-based Architecture) and the relationship among them.

At first, based on the policy, the smart grid system determines that it needs to adapt, and through the defined relationship between *Warning* state and *Demand Bidding* goal in APDO, *Demand Bidding* goal is selected as the proper goal, which are the answers of the first and second questions in Figure 3. *Demand Bidding* goal has *satisfy* relations with *Recruit Participant*, *Plan to Support Electric Rate*, *Construct Load Control Plan* and *Control Manage Systems* behavior features. These relations support that these four behavior features become the answer of the third question. Using act relation between behavior feature and component feature, *Customer Information System*, *Distribution Management Systems*, *Energy Management*, and *Demand Response Management System* are selected as the appropriate component features. It is the answer of the fourth question in Figure 3. In the fifth and sixth questions, these component features are converted to the roles and organization shown in the bottom of Figure 3 via *convert* relation between them, and these roles or organizations are played by the smart grid system components capable to perform them to change its architecture and satisfy the new goal. Consequently, during the adaptation process, these decisions are addressed by answering the questions shown in Figure 3 through APDO and the system changes its goals, features, and architecture [13].

Using the proposed adaptation process, the system makes an appropriate decision with convincing evidences to assure the high quality of the adaptation based on the answer to decision questions. It also support the stakeholders in understanding the rationale behind the adaptation, as they are able to know why the adaptation happens and how it is processed.

IV. Related Works

In related work, we examine existing self-adaptive system frameworks. We will compare other frameworks with the one proposed in this paper, especially in terms of decision making in the adaptation process and knowledge representation.

Rainbow is a framework for developing customized self-adaptive system [10]. It is composed of two components: managed system and manage system. Managed system is the system, which directly adapts to the environment. Manage system controls managed system through MAPE-K process. In rainbow, the strategies are defined as the adaptation unit, which a system can take. A strategy contains the system architecture and several tactics. And, it is defined through Stitch and Acme. Using utility theory, a system is able to quantify which strategy can achieve system objective with the highest utility value. Based on these results, a system selects and changes its architecture that is suitable for new situation.

As mentioned before, the adaptation unit for rainbow is strategy. All the strategies that the system can take are already defined at design time. Thus, a system cannot consider various adaptation problems and provide enough flexibility of the system architecture. However, proposed approach models only knowledge for adaptation and a system infers the appropriate decision using that knowledge at run-time. In other words, the proposed approach does not determine the available architecture, but design knowledge in order to determine appropriate architecture at run-time. It supports high flexibility and traceability by providing the evidence of the adaptation. Proposed approach provides high understanding of the adaptation process to the stakeholders as well.

MADAM (Mobility and Adaptation enabling Middleware) is specially focused on the middleware for the self-adaptation at mobile platform [11]. Through MDA (Model-Driven Architecture), the user defines the system architecture model and the system adapts to new situation by changing the architecture model. In order to select the most suitable architecture model, MADAM uses parameterization, which is a method to apply the external variables to the predefined adaptation formula. The adaptation is processed through functionalized decision making process which means that the situations which the system can face are mapped one-to-one with each architecture model.

MADAM has adaptation middleware to manage system architecture and adaptation process. Thus, the engineer defines this middleware at design time. This adaptation is performed by predefined mapping knowledge, therefore MADAM is not able to consider run-time perspective in the adaptation such as constructing new architecture model, which is more suitable than other defined architecture model. In the proposed framework, we refer MDA approach to represent system architecture with various abstract level, but we infer the adaptation result through knowledge at run-time in order to make an appropriate decision. Namely, we define no direct solution for each situation, but provide knowledge to support decision making process and decide the solution for the system.

DiVA (Dynamic Variability in complex, Adaptive systems) is the framework to support developing the self-adaptive system using AOP (Aspect-oriented Programming) [12]. They use base model and aspect model as the adaptation units for the system adaptation. The base model is designed from the essential components and the aspect model is designed based on the optional component that is able to be added or modified. Simultaneously using both the models, the engineer can easily design the system variability and consider various situations. At design time, not only base and aspect model, but

dependency between aspects in variable points, policy and context are defined as well. Each context and policy is connected to the available aspects, and the system is weaving with base model and selected aspect models at runtime to construct complete system architecture.

In DiVA, the adaptation units are defined at design time as aspect models and it constructs complete system architecture using these models at run-time. It is impossible that a system uses undefined and new aspect for comprising new system architecture. Therefore, it is impossible to consider semantic dependencies when new dependency is defined or many aspects are intertwined. However, proposed approach can manage not only the syntactic relation, but also the semantic relation through ontology by defining knowledge and reasoning the semantics behind that knowledge. It also assures that a self-adaptive system can make a more appropriate decision.

V. Conclusion and Future Works

In this paper, we propose a knowledge-intensive software engineering framework for self-adaptive systems. The proposed framework supports the decision making process and the traceability of the adaptation knowledge through knowledge-intensive inference and questions. Thus, the system engineer and stakeholders are able to comprehensively understand the adaptation process and analyze the problem and solution to change non-adaptive systems to be self-adaptive. The limitation of software adaptability is mitigated by extending ontology to add new knowledge for the needed adaptation such as new models or emergent relationships between existing models.

In future works, we need to define an ontology-based software development methodology. In this methodology, the process of knowledge construction about the domain and target system, and the fundamentals for self-adaptation should be defined. Also, inference in decision making process should be extended to resolve uncertainty problems. Uncertainty is an emergent issue in the self-adaptive system. Lastly, the verification and validation of the adaptation framework are needed in order to determine the correctness of knowledge and the decisions made during the adaptation.

Acknowledgment

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No. 2013M3C4A7056233).

References

[1] Betty H. Cheng et al., Software Engineering for Self-Adaptive Systems: A Research Roadmap, Software Engineering for Self-Adaptive Systems, Springer, 5525, Lecture Notes in Computer Science, 2009, 1-26.

[2] IBM, An architectural blueprint for autonomic computing, Autonomic Computing White Paper, 2006

[3] De Lemos, Rogério, et al. "Software engineering for self-adaptive systems: A second research roadmap." Software Engineering for Self-Adaptive Systems II. Springer Berlin Heidelberg, 2013. 1-32.

[4] Kephart, Jeffrey O., and David M. Chess. "The vision of autonomic computing." Computer 36.1 (2003): 41-50.

[5] Yijun Yu, et al. 2008, From Goals to High-Variability Software Design, Foundations of Intelligent Systems, Springer, 4994, Lecture Notes in Computer Science (2008), 1-46.

[6] Brice Morin, et al. 2009. Taming Dynamically Adaptive Systems using models and aspects. In Proceedings of the 31st International Conference on Software Engineering. IEEE Computer Society, Washington, DC, USA, 122-132.

[7] Nelly Bencomo, et al. 2008, Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems, 2nd International Workshop on Dynamic Software Product Lines.

[8] Seedorf, Stefan. "Applications of ontologies in software engineering." In 2nd International Workshop on Semantic Web Enabled Software Engineering held at the 5th International Semantic Web Conference. 2006.

[9] Gruber, Thomas R. "Toward principles for the design of ontologies used for knowledge sharing?" International journal of human-computer studies 43.5 (1995): 907-928.

[10] David Garlan, et al. "Rainbow: architecture-based self-adaptation with reusable infrastructure," Computer, vol.37, no.10, pp. 46- 54, Oct. 2004

[11] Sebastiano Lombardo, "D.8.9: Mobility and Adaptation enabling Middleware: Final Report", MADAM final report, 2007

[12] DiVA Project Consortium, "D7.4: A Model-based Approach for Construction and Run-time Management of Adaptive Systems: DiVA practices and Lessons Learned", DiVA White Paper, 2011.

[13] Brice Morin, et al., 2009, "Models@ Run.time to Support Dynamic Adaptation", IEEE Computer, 42, 10, 2009, 44-51.

[14] Goldsby H.J. et al. 2008. Goal-Based Modeling of Dynamically Adaptive System Requirements. International Conference on Engineering of Computer-Based Systems.

[15] Alexei Lapouchnian, Sotirios Liaskos, John Mylopoulos, Yijun Yu, 2005, Towards requirements-driven autonomic systems design, Proceedings of the 2005 workshop on Design and evolution of autonomic application software

[16] Kang, Kyo C., et al. Feature-oriented domain analysis (FODA) feasibility study. No. CMU/SEI-90-TR-21. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1990.

[17] Capilla, Rafael, Jan Bosch, and Kyo-Chul Kang. "Systems and Software Variability Management." pp. 25-32, 2013

[18] Colman, Alan Wesley. "Role oriented Adaptive Design". Swinburne University of Technology, Faculty of Information & Communication Technologies, 2006

[19] Oreizy, Peyman, et al. "An architecture-based approach to self-adaptive software." Intelligent Systems and Their Applications, IEEE 14.3 (1999): 54-62.

[20] Colman, Alan, and Jun Han. "Roles, players and adaptable organizations." Applied Ontology 2.2 (2007): 105-126. Korea Power Exchange, "Power market operating rule", 2013

[21] Colman, Alan, and Jun Han. "Using role-based coordination to achieve software adaptability." Science of Computer Programming 64.2 (2007): 223-245. Korea Power Exchange, "Power market operating rule", 2013

[22] Lee, S.W. and Gandhi, R. A., Ontology-based Active Requirements Engineering Framework, In Proceedings of the 12th Asia-Pacific Software Engineering Conference, 2005. IEEE Computer Society

[23] Daniel M. Berry, Betty H. C. Cheng, Ji Zhang, 2005, The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems, In 11th International Workshop on Requirements Engineering Foundation for Software Quality.

[24] U.S Department-of-energy. "Grid 2030: a national Vision for electricity' second 100 years". Tech. report, Department of energy, 2003

[25] Korea Power Exchange, "Power market operating rule", White Paper, 2013