IEEE SysCon 2009 —3rd Annual IEEE International Systems Conference, 2009
Vancouver, Canada, March 23–26, 2009

Requirements Engineering for Systems of Systems

Grace A. Lewis, Edwin Morris, Patrick Place, Soumya Simanta, Dennis B. Smith
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
{glewis, ejm, prp, ssimanta, dbs}@sei.cmu.edu

***Abstract*—Traditional requirements engineering for single systems, while remaining a large challenge for engineers, has been extensively researched and many techniques have been proposed and used with varying degree of success. However, many modern systems of systems are being developed to support interaction across multiple controlling authorities and existing techniques are proving to be inadequate for meeting the challenges of requirements engineering for systems of systems. This paper discusses some of these challenges, examines several existing techniques, and discusses how these techniques could be applied to engineer requirements for systems of systems.**

Keywords- Systems of Systems, SoS, Requirements Engineering, System Requirements, Software Requirements

I. INTRODUCTION

Fred Brooks in a seminal paper wrote *"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is so difficult as establishing the detailed technical requirements. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later"*[1]. It is now widely accepted that Brooks was correct—requirements engineering is one of the most critical and difficult tasks in developing any system.

Traditional requirements engineering for a single system, while remaining one of the most vexing challenges for engineers, has been extensively researched and many techniques have been proposed and used with varying degrees of success. However, because many modern systems of systems are being developed to support interaction across multiple controlling authorities (i.e., multiple system owners, operators, etc.), existing techniques are proving to be inadequate. As a result, requirements engineering becomes an even greater challenge for multiple interacting systems that form a system of systems (SoS) [2-4].

Techniques such as Goal-Oriented Requirements Engineering (GORE) [5, 6]; scenario-based requirements engineering [7]; Variability, Commonality and Scope (VCS) analysis [8, 9]; Capability Engineering [10]; and Design Structure Matrix (DSM) [11] have been successfully used to engineer systems. Each of these techniques supports several (although not necessary all) requirements engineering activities that typically include elicitation, analysis, modeling, refinement, specification, and documentation. This paper focuses on the "early" requirements activities of elicitation, modeling, and analysis that include what we consider to be the fundamental problems of identifying and understanding requirements. "Later" activities that are focused on communicating and documenting requirements are not addressed.

This paper discusses the challenges of requirements engineering in an SoS context. It then examines several existing requirements engineering techniques and discusses a proposal for applying these techniques to a process for engineering requirements for systems of systems.

II. REQUIREMENTS ENGINEERING CHALLENGES FOR SOS

Individual systems are becoming increasingly complex. They have many stakeholders and are intended to be deployed in diverse environments. However, several factors complicate the requirements engineering process in the SoS context. These factors are identified in the following sections.

A. Scale

Issues of scale such as number of contexts, multiple interactions among constituent systems, and large numbers of end users and stakeholders to be supported make the understanding of requirements for SoS more complex [2, 12]. These factors increase the flexibility required of SoS constituents while also increasing the number of constraints imposed on the engineered solution.

B. Multi-Domain

Unlike individual systems, SoS often cross domains. That is, the constituent parts (i.e., the individual systems) of an SoS can belong to different domains. For example, while healthcare, insurance, retail, tax and government systems may each support *customers*, the concept of "customer" varies widely across these domains. An SoS linking healthcare, government, and insurance systems together must rectify the domain-specific meanings of customer. Also, requirements for individual capabilities (e.g., credit card processing or security mechanisms such as single sign-on) must reflect the fact that the capability may be used in widely different domains (e.g., the same capability of credit card processing can be used in healthcare, retail, and banking).

C. Varied Operational Context

Even within a single domain, specific operational situations in which the capability will be used can vary. For example, the demands placed on an Electronic Medical Record (EMR) system vary depending on whether the system will be used inside a hospital (i.e., inpatient care) or in a doctor's office (i.e., outpatient care). Capabilities provided by an EMR system that must be accessible from

both a hospital and a doctor's office are likely to become part of different workflows or business processes depending on the context.

D. Decentralized Control

Even within a single domain and context, consensus must be established across multiple interested and affected parties. Typically there is more than one decision-making authority controlling various aspects of the SoS [2-4]. Moreover, in case of conflicting needs each constituent system tries to ensure that it satisfies its own requirements first [3].

If the constituent systems of an SoS are controlled by different organizations, or even geographically distributed within the same organization, then is it difficult for a single team to understand all the requirements of an SoS at all levels. In this case, requirements engineering has to be performed through collaboration of multiple stakeholders during a long and continuous development and evolution phase.

E. Rapidly Evolving Environments

Changes in technology, unpredictable and changing stakeholder demands, new and modified laws and regulations, updated business processes of interconnected systems, and many other factors outside the control of the any single participating system can lead to rapidly changing demands on an SoS and its constituent systems [10, 13, 14]. Such changes also affect individual systems, but in an SoS, single changes often cascade through multiple systems. Because there are so many more opportunities for change (i.e., at each individual system), change becomes the normal state in an SoS.

F. Continuous and Often Disconnected Execution of Multiple Life-Cycle Phases

Most SoS do not have a single, sequential life cycle that synchronizes all of the individual systems (i.e., a unified sequence of requirements, design, implementation, integration, testing, deployment, maintenance, and decommissioning) [2]. Each constituent system of the SoS frequently follows its own life cycle, and a large SoS may have several constituents at any given point in their life cycles. This presents a difficult problem from a requirements engineering perspective because some constituent systems may be at an early point in their development and have highly unstable requirements, while other constituent systems may be highly stable but near decommissioning, and thus represent a point of future volatility. In order to enforce synchronization among individual, the U.S. Army uses the concept of software blocking [http://www.sec.army.mil/secweb/value_added/software_blocking_vas.php]. However, they are finding it difficult to deploy blocks given some of the challenges just mentioned plus other challenges [15]. Thus, the difficult task of the requirements analyst is to understand what capabilities are currently available, what capabilities will be available in the future, and what capabilities will disappear in the near future.

G. Opportunistic Needs to Collaborate and Integrate

An SoS often consists of socio-technical systems, which involve and depend on complex interactions among systems and people. Solutions to existing problems in many social and technical domains can be automated using technology i.e., human-human and human-machine interactions can be automated as machine-machine interactions. Sometimes opportunities for automation are not recognized until some level of automation is introduced. As more and more systems are introduced, new opportunities are recognized. These new demands not only require existing systems to adapt to provide newer capabilities but also demand that new computer systems be created to help or take more responsibilities from humans.

III. Goals of SoS Requirements Engineering

While we believe that existing requirements engineering techniques commonly applied to single systems are inadequate for SoS, it would be foolish to ignore important insights that can be gained from these techniques. Existing techniques, such as the ones mentioned earlier, can either provide activities to meet SoS requirements engineering goals or could be fully adapted to work in SoS contexts.

At a high level, the goals of SoS requirements engineering are:

1. Consideration of the perspective of the individual systems as well as the SoS
   - Identification of capabilities that are expected from the entire SoS. For example, consider the identification of capabilities and goals of a network of regional healthcare providers forming a Regional Health Information Organization (RHIO).
   - Identification of independent needs of individual systems (current and future) that constitute the SoS. For example, engineering an EMR system for a hospital requires understanding the requirements that are specific to the hospital needs for inpatients. Existing requirements elicitation techniques (e.g., use cases) can be used to identify requirements for the EMR system. However, if the EMR system is part of a bigger healthcare infrastructure, then these techniques are not sufficient. Elicitation of requirements and expectations from other interconnected systems and the operational environment of the EMR systems becomes equally important.
2. Consideration of SoS current and future needs against individual capabilities
   - Understanding how best to engineer individual constituents systems so that they not only meet the requirements of interoperating with existing systems but also are flexible enough to meet future demands with the minimum amount of change
   - Understanding how existing capabilities provided by individual systems can be combined to meet the goals of the SoS. This requires an analysis of the gap

between requirements placed on the SoS and the capabilities provided by individual participating systems.

- Understanding current and future demands on the SoS environment so that adaptable and robust solutions can be engineered. Multiple stakeholders and variability in operating contexts are some key challenges for satisfaction of this goal.

IV. Requirements Engineering Approach For SoS

Successful requirements engineering for an SoS requires a combination of top-down and bottom-up approaches. A top-down approach used in isolation does not adequately consider the needs of individual systems, and an isolated bottom-up approach will not capture the essential expectations for the SoS as a whole. The top-down aspects include understanding the SoS type, the environments associated with the SoS and its constituent systems, the identification of goals, and the identification of SoS interactions. The bottom-up aspects include understanding the capabilities provided by individual constituent systems and the internal and external constraints placed on these systems. What follows is a summary of an SoS requirements engineering approach that considers these top-down and bottom-up aspects and incorporates activities from existing requirements engineering approaches.

A. Identify SoS Context

The first step in the requirements engineering of an SoS is to identify its context: the type of SoS and the environments related to the SoS and the constituent systems.

1) SoS Type

From an engineering perspective, a relevant classification of SoS is provided in the DoD Systems of Systems Engineering Guide [3]. This guide classifies SoS as directed, acknowledged, collaborative, or virtual, in which operational control gradually decreases from first to last. Operational control refers to centralized management of the technical and non-technical aspects of constituent system development, deployment and execution. Operational control is an important attribute from an SoS requirements perspective because it captures the extent to which the requirements for the individual systems are centrally defined by the SoS. For example, if there is a clear mandate and authority controlling all aspects of a RHIO then it becomes a directed SoS in which the RHIO places requirements on all individual systems, even if the individual systems maintain the ability to operate independently. In the case of an aircraft or an automobile that could be considered acknowledged SoS during their development, the constituent systems retain their independence, but requirements (and changes in requirements) are based on collaboration between the SoS and the individual systems. In the case of collaborative and virtual SoS this centralized control is minimized and therefore requirements engineering is based on less concrete requirements that require individual systems to be built with flexibility and adaptability un mind (on top of all the functional capabilities) and for SoS to be more "open-minded" to work with existing capabilities as opposed to dictating requirements for individual systems. From these examples it should be clear that identifying the SoS type should be the first step of SoS requirements engineering because it dictates the shape that the requirements will take for both the individual systems as well as the SoS.

2) SoS and System Envronments

Each constituent system, and the SoS itself, is influenced by its environment [16]. For example, lab systems, pharmacies, and EMRs are participating systems in a RHIO, but also have their individual, unique decision support rules, governance organizations, and regulatory statutes. These and other aspects of the environment that must be considered include

- **Entities:** The environment consists of entities that may not be directly related to the SoS or the systems but do exert considerable influence over them [13]. For example, a RHIO's environment consists of entities such as healthcare standard organizations (e.g., HL7), legislative and regulatory bodies (e.g., department of health), policy-making bodies, emerging technologies (e.g., semantic technologies for healthcare), and the healthcare research community.
- **Influence:** Environmental entities have varying degrees of influence on the systems and the SoS. For example, if a majority of healthcare software vendors decide to adopt a new technology (e.g., Service-Oriented Architecture (SOA)) then the remaining vendors must consider a similar move to stay competitive. This may eventually cause individual systems to adopt the new technology and potentially influence the technologies used to implement the RHIO.
- **Evolution:** Understanding the elements of the environment and their influences on the SoS is not enough because the environment is dynamic (i.e., the entities and their influences are changing) [14]. For example, healthcare systems implement standards and are affected by how these standards evolve. It is not only important to understand changes in the environment but also the rate and type of change. If a new version of a standard is released every couple of years, then systems must be designed to accommodate these changes with minimum amount cost and effort.

Understanding the environment helps individual systems and the SoS anticipate future needs and engineer appropriate mechanisms (e.g., architectural) to adapt to these changes in the environment.

B. Identify SoS and Individual System Goals

The next step is the identification of goals of the SoS and of the individual participating systems. GORE [5, 6] is a popular approach for systems requirements elicitation that can be used successfully to model goals in an SoS context. In this approach, stakeholder needs are captured as goals. Goals can be captured at various levels of abstraction, from high-level strategic goals to low-level technical goals. High-level

goals are refined and modeled as AND-OR goal trees. Goals can represent both functional and quality expectations. An example of a high level functional goal for a RHIO could be that electronic patient information should be available to any healthcare provider in a geographic region. Additionally, a quality goal could require patient information to be shared securely and without violating the patient's privacy. Creating a goal tree for an SoS and for the individual systems is possible especially in the case of directed and acknowledged SoS because of the existence of a global top-down view of SoS that includes the participating systems. For example, in case of a RHIO, some entity is responsible for understanding the overall goals of the RHIO (SoS) and coordinating multiple participating healthcare providers (systems). Analysis of these goal trees can provide insights into common and conflicting goals.

1) Common Goals

Goal trees can provide insight into the common goals for multiple systems. While commonality of goals at a high level is trivial, their value is in the identification of goals at every level including technical goals and goals belonging to different categories (e.g., functional, quality attribute related) [6]. Each lower-level goal is implemented by a software system or a set of systems. Therefore, identification of goals from an SoS requirements perspective allows the identification of systems that can implement or provide similar capabilities. Concepts from commonality analysis can be used to identify common goals across goal trees of multiple participating systems [8]. For example, in a RHIO, a common technical goal for multiple healthcare providers could be a capability to convert between data formats.

2) Conflicting Goals

Goal trees can also be valuable in the identification of conflicting goals. For an SoS, they can be used not only to identify conflicting goals between participating systems but also between an SoS and the participating systems. It is important to identify and understand these conflicts because individual systems often give higher priority towards meeting their individual goals. Therefore, if an SoS relies heavily on individual systems to provide the necessary capabilities it must find appropriate resolutions for these conflicts with the individual systems.

C. Understand SoS Interactions

In an SoS, capabilities provided by participating systems interact to provide SoS capabilities. One useful view of an SoS is as a set the interactions that the SoS must support. Additionally, viewing an SoS as a set of interactions makes it easier to engineer individual interactions. An example of an interaction in a RHIO is "view patient information". A clinician, irrespective of physical location in the RHIO, should be able access a patient's health record even if various parts of the patient record are stored and controlled by different participating healthcare providers. For this to be possible, each healthcare provider must provide capabilities to locate and access patient information under their control and the SoS should be able to integrate these capabilities to provide a unified patient record.

Identifying SoS interactions enables better comprehension of the SoS because they present a top-down view of the SoS. Each SoS interaction should be mapped to the SoS goal tree, explaining how SoS goals are satisfied by SoS interactions [17].

Identifying important interactions associated with the SoS and understanding an interaction's functional and quality requirements is an approach to requirements engineering for an SoS. An SoS cannot meet its goals successfully unless its interactions are implemented successfully. Interactions are composed of sub-interactions that can be machine-to-machine, machine-to-human, and human-to-human in nature. Understanding an interaction requires identification of these sub-interactions.

Scenario-based requirements engineering techniques [7], use cases, and change cases [13] are some of the techniques that can be used to extract SoS interactions. Participating systems' capabilities can interact in multiple contexts as discussed in the following subsections.

1) Process-Centric Interactions

The first type of interaction represents a process or a workflow supported by the SoS. Process-centric interactions involve flow of data and control across boundaries of multiple participating systems. The processes are implemented by threading capabilities provided by participating systems. For example, the "medical bill payment" process may potentially involve many healthcare systems and providers such as a billing system used by the doctor's office, an insurance claim system, a lab system, and a credit card processing system. It may not be possible to automate all elements of the process because some required capabilities may be missing. From an requirements engineering perspective, process modeling techniques can be applied to model these processes at various levels. For example, if the SoS instance is service-oriented, the processes can be modeled using an modeling and execution language such as the Business Process Execution Language (BPEL).

2) Data-Centric Interactions

Another common interaction in an SoS is data-centric, where data from multiple systems is processed (e.g., aggregated, filtered, transformed, and translated) to support SoS needs that cannot be met by any single system. Mashups [18] are popular examples of data-centric interactions in an SoS context. Both data-centric and process-centric interactions are useful in supporting functional SoS capabilities.

3) Resource-Centric Interactions

Resources controlled by various participating systems can be shared or pooled to provide capabilities to the SoS. Sharing of computing resources, such as CPU cycles and disk space in Grid and Cloud Computing [19] environments, is an example of a resource-centric SoS interaction. For example, RHIO-specific storage space could be a federation of disk space provided by participating healthcare systems. Resource interactions are commonly used to support SoS infrastructure capabilities.

D. Identify Individual System Capabilities and Constraints

SoS capabilities cannot be successfully composed without a proper understanding of the capabilities provided by individual systems. The bottom-up analysis involves identification of capabilities of constituent systems. This activity can be performed in parallel with the top-down activities and should provide valuable information to understand SoS interactions.

1) Individual System Capabilities

Identification of actual capabilities needs to be performed for existing or planned individual systems. The goal of this activity is to identify coarse-grained functionality and features provided by individual systems in terms of inputs required, outputs expected and the constraints under which the capability works correctly. An example capability provided by a clinical lab system is a set of interfaces that accept lab orders (input) and return lab results (output) in a standard format (e.g., HL7). A starting point for identification of system capabilities can be a goal tree for the system. A list of scenarios supported by the system can also help identify the capabilities supporting these system-specific scenarios.

2) Capability Constraints

A system capability may have several constraints associated with it. For example, if a capability exists in a legacy system and is implemented using an outdated technology or if the capability cannot be accessed outside a security enclave (e.g., a secure military network). Understanding of these constraints is important from a requirements engineering perspective because they can hinder and in some cases make it impossible to use the capability.

E. Analyze the Gap

Top-down and bottom-up analysis should be followed by a gap analysis. The main goal of this activity is to reconcile outputs from the top-down and bottom-up analysis and to identify gaps in terms of capabilities currently not provided by existing systems and capabilities that require modifications to support SoS goals.

1) Goal-Capability Mapping

Mapping goals from system trees to individual capabilities is a mechanism to identify individual capabilities that are either already available or are currently being developed. Creating a goal-capability mapping has several advantages. First, goals provide a good starting point for identifying capabilities at system level. Second, the capabilities can be identified at the same level of granularity as modeled in the goal tree. Finally, the goal-capability mapping relates the problem domain to the solution domain. It facilitates traceability and completeness to capability identification. The system capabilities identified in this step will be used to realize SoS interactions.

2) Capability Dependency

Once the system capabilities have been identified, the next step is to identify dependencies across capabilities. Capability dependencies can exist within a single system or across multiple systems. DSM is a useful tool to model these dependencies and identify coupling between capabilities [10, 20]. If a system capability is used by an SoS, it is essential to understand the effect of changes on the system capability being used. For example, if the capability used in an SoS interaction has high coupling, it means any changes to other capabilities inside the system may result in changes to the capabilities being used.

3) Capability Reengineering

Some system capabilities require reengineering before they are ready for external use. For example, a capability may not have security mechanisms because it was designed for internal use only. Changes to existing capabilities may be expensive or could break the current system. Reengineering of capabilities should be done to maximize their reuse. This can be achieved by identifying requirements for modifications to capabilities such that 1) the existing systems have the least impact due to changes, 2) the capability has the appropriate granularity, and 3) the capabilities have high cohesion and low coupling [10, 20]. These attributes maximize the reusability of the reengineered capabilities.

Gap analysis should identify the type of changes that would be required to support SoS interactions. There are several types of modifications that can occur during capability reengineering—granularity, quality of service and functionality. Understanding capabilities in the context of current and future SoS interactions can provide valuable information for designing these capabilities.

4) Missing and New Capabilities

An important part of gap analysis is identification of capabilities not provided by constituent systems. The goal-capability mapping can be used to create an initial list of missing capabilities—goals that do not map to any capabilities. Missing capabilities can be addressed by looking at additional systems that provide those capabilities or by developing new capabilities. These new capabilities should be designed to be as change-tolerant as possible.

Not all capabilities can be provided by constituent systems. In some cases, capabilities may not be currently required by individual systems but only by the SoS. For example, an ability to compose capabilities from multiple systems may not exist in any single system. However, such a capability will be required by almost all SoS interactions. It is important to carefully identify requirements for such capabilities and engineer them to be reusable in multiple SoS interactions.

Finally, if the goal of the SoS is to minimize new development, the outcome of the gap analysis activity could be the realization that SoS goals may have to be adjusted in order to use existing capabilities.

V. Summary and Conclusions

At a high level, requirements engineering approaches for SoS need to consider the perspective of the individual systems as well as the SoS perspective. In addition, SoS current and future need to be evaluated against individual system capabilities. These considerations require a

combination of top-down and bottom-up approaches for requirements engineering.

The SoS requirements engineering approach proposed in this paper includes top-down and bottom-up activities:

1. Identify SoS context
2. Identify SoS and individual system goals
3. Understand SoS interactions
4. Identify individual system capabilities and constraints
5. Analyze the gap

While we believe that existing requirements engineering techniques commonly applied to single systems are inadequate for SoS, it would be foolish to ignore important insights that can be gained from these techniques. The next steps in this work are to continue exploring existing requirements engineering approaches to identify techniques that can be applied directly or adapted to work in SoS contexts. Also, as the technologies to support virtual types of SoS mature, it will become important to validate the applicability of the proposed approach and to complement it with techniques that work in a more decentralized, multi-organizational context.

REFERENCES

[1] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, 2 ed.: Addison-Wesley Professional, 1995.

[2] G. Lewis, E. Morris, P. Place, S. Simanta, D. Smith, and L. Wrage, "Engineering Systems of Systems," in *Systems Conference, 2008 2nd Annual IEEE*, 2008, pp. 1-6.

[3] Office of the Deputy Under Secretary of Defense for Acquisition and Technology and Systems and Software Engineering, "Systems Engineering Guide for Systems of Systems," ODUSD(A&T)SSE, Ed., 2008.

[4] M. W. Maier, "Architecting Principles for Systems-of-Systems," *Systems Engineering,* vol. 1, pp. 267-284, 1999.

[5] A. van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour," in *Fifth IEEE International Symposium on Requirements Engineering*, 2001, pp. 249-262.

[6] A. I. Anton and C. Potts, "The Use of Goals to Surface Requirements for Evolving Systems," in *Proceedings of the 1998 International Conference on Software Engineering*, 1998, pp. 157-166.

[7] A. Sutcliffe, "Scenario-based Requirements Engineering," in *Proceedings of 11th IEEE International Requirements Engineering Conference*, 2003, pp. 320-329.

[8] J. Coplien, D. Hoffman, and D. Weiss, "Commonality and Variability in Software Engineering," *Software, IEEE,* vol. 15, pp. 37-45, 1998.

[9] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, 3rd ed.: Addison-Wesley Professional, 2001.

[10] R. Ramya, D. A. James, and A. B. Shawn, "Capabilities Engineering: Constructing Change-Tolerant Systems," in *40th Annual Hawaii International Conference on System Sciences (HICSS)*, 2007, pp. 278b-278b.

[11] T. R. Browning, "Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions," *IEEE Transactions on Engineering Management,* vol. 48, pp. 292-306, 2001.

[12] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau, "Ultra-Large-Scale Systems: The Software Challenge of the Future.," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 2006.

[13] J. Earl F. Ecklund, L. M. L. Delcambre, and M. J. Freiling, "Change Cases: Use Cases That Identify Future Requirements," in *Proceedings of the 11th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications* San Jose, California, United States: ACM, 1996.

[14] K. Knoll and S. L. Jarvenpaa, "Information Technology Alignment or Fit in Highly Turbulent Environments: The Concept of Flexibility," in *Proceedings of the 1994 computer personnel research conference on Reinventing IS : managing information technology in changing organizations.* Alexandria, Virginia, United States: ACM, 1994.

[15] B. C. Meyers, J. D. Smith, P. Capell, and P. R. H. Place, "Requirements Management in a System-of-Systems Context: A Workshop," Software Engineering Insititute, Carnegie Mellon University, Pittsburgh, PA CMU/SEI-2006-TN-015, 2006.

[16] G. M. Weinberg, *An Introduction to General Systems Thinking*, 25 Anniversary Edition ed.: Dorset House Publishing Company, Incorporated, 2001.

[17] B. List and B. Korherr, "An Evaluation of Conceptual Business Process Modelling Languages," in *Proceedings of the 2006 ACM symposium on Applied computing* Dijon, France: ACM, 2006.

[18] J. Wong and J. I. Hong, "Making Mashups with Marmite: Towards End-user Programming for the Web," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* San Jose, California, USA: ACM, 2007.

[19] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Grid Computing Environments Workshop, 2008. GCE '08*, 2008, pp. 1-10.

[20] L. Jong Kook, J. Seung Jae, K. Soo Dong, J. Woo Hyun, and H. Dong Han, "Component Identification Method with Coupling and Cohesion," in *Eighth Asia-Pacific Software Engineering Conference (APSEC)*, 2001, pp. 79-86.