A Study of Resilient Architecture for Critical Software-Intensive System-of-Systems (Sisos)

Nadeem Akhtar
Department of Computer Science & IT
The Islamia Univ. of Bahawalpur, Pakistan

Malik Muhammad Saad Missen
Department of Computer Science & IT
The Islamia Univ. of Bahawalpur, Pakistan

Nadeem Salamat
Department of Basic Science and Humanities
Khawaja Fareed Univ. of Engineering and Technology, RYK, Pakistan

Amnah Firdous
Department of Computer Science
COMSATS Institute of Information Technology, Pakistan

Mujtaba Husnain
Department of Computer Science & IT
The Islamia Univ. of Bahawalpur, Pakistan

***Abstract*—The role of critical system-of-systems have become considerably software-intensive. A critical system-of-system has to satisfy correctness properties of liveness and safety. As critical system-of-systems have to operate in open environments in which they interact and collaborate with other systems, satisfy action of the requirements through traditional offline top-down engineering no longer suffice. Most of the critical software-intensive system-of-systems have no fixed boundaries and services provided by other systems will come and go in unpredictable ways; in these systems assuring correctness is a challenging issue. These systems need to tolerate faults in the face of change; they need a resilient architecture. An approach has been proposed for the analysis, design, formal specification and verification of critical Software-intensive System-of-Systems.**

***Keywords*—Resilient architecture; Critical systems; System-of-System (SoS); Software-intensive SoS (SiSoS); Emergent behavior; Correctness; Safety**

I. Introduction

Most of the critical Software-intensive System-of-Systems (SiSoS) have no fixed boundaries (i.e. they have open environment); and services provided by other systems come and go in unpredictable ways. In such system-of-systems assuring correctness by construction is not possible. Such system-of-systems need to tolerate faults in the face of change; in short they need to be resilient.

Nowadays, software systems are performing critical tasks, thus more and more software systems are becoming critical. Defects in a critical system can cause human life loss, and can also have a dramatic impact on the environment. The functions performed by these systems have become considerably software-intensive. The software of critical systems has to satisfy correctness properties of *liveness* and *safety*.

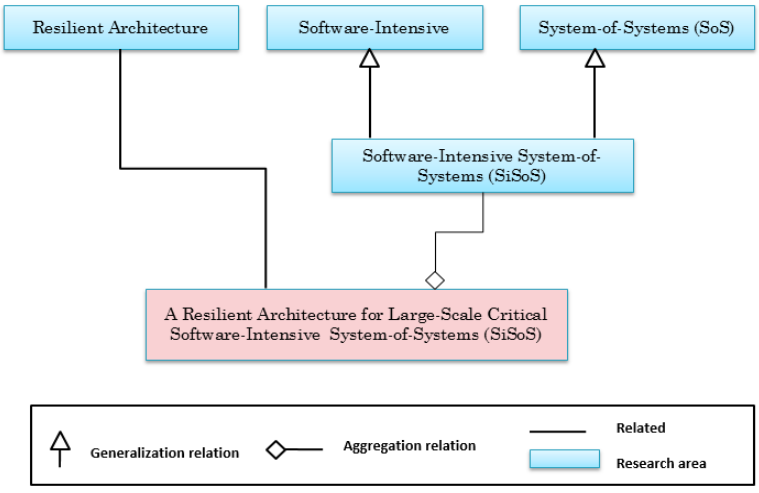

Fig. 1. Study domain

As critical systems increasingly have to operate in open environments in which they interact and collaborate with other systems, satisfaction of the requirements through traditional offline top-down engineering no longer suffice

A. Software-Intensive System-of-Systems (SiSoS)

A system is a collection of elements that work together and produce results that cannot be obtained by the elements operating individually. An individual element of a system may itself be large and complex, and comprised of sub-elements acting in concert with one another.

A System-of-System (SoS) integrates independently useful systems into a larger system, delivering new unique functions to users that emerge from the combination of the individual parts. Examples are *intelligent traffic systems*, *integrated surveillance systems*, and *networked smart homes*. Engineering SoS and guaranteeing runtime qualities (i.e. *performance*, *reliability* etc.) is complex due to a variety of uncertainties. Examples of such uncertainties are *systems that attach and detach at will*, *dynamically changing availability of resources*, and *faults and intrusions that are difficult to predict*.

SoS are formed by the integration of autonomous and heterogeneous systems. The SoS were first applied in the analysis and design of military systems of the American Department of Defense. [3]

SoS is used as a method to reach goals or provide unique capabilities for the collaborative work between existing systems. [4][5]

The first definitions and taxonomies for SoS were introduced by Maier [4] in 1990's in which three SoS basic types (*virtual*, *collaborative*, and *directed*) are proposed. He also specified the five characteristics (*operational independence*, *managerial independence*, *evolutionary development*, *emergent behavior* and *geographic distribution*) of SoS. Based on this characterization, Maier identifies a set of guiding design principles for SoS:

- *Stable intermediate forms:* The individual systems or subsets of systems of a SoS should be capable of operating and fulfilling useful purposes, before full deployment and during operation.
- *Policy triage:* SoS design team should carefully choose what to control; over-control will fail for lack of authority, under-control will eliminate the integrated nature of the SoS.
- *Leverage at the interfaces:* The architecture of SoS is essentially defined by its interfaces, which are the primary points at which designers can exert control.
- *Ensuring collaboration:* Mechanisms should be exploited that create joint utility, which is known to be a basis for consistent behavior.

[6] refers to SoS or Federations Of Systems (FOS) or Federated Systems Of Systems (F-SOS) as systems that possess characteristics of complex adaptive systems. [7] focuses on the nature of the composition to define the distinguishing characteristics of SoS, including *autonomy*, *connectivity*, *diversity* and *emergence*. [8] stresses scale and complexity as central properties of ultra-large scale systems, phrased by the slogan *"scale changes everything"*. [9] describes SoS as a combined arrangement of managerially independent and geographically distributed elements (i.e. already fulfilling some purposes) put together to work and provide a functionality that is not possible otherwise.

Energy systems, healthcare systems, logistic systems, and transportations systems can be designed and developed based on SoS concept. [10]

SoS are complex and large-scale systems and are software-dependent, therefore they become Software-Intensive SoS (SiSoS). [11]

SoS facilitates development of complex systems. It is a composition of systems in which its constituents are themselves systems. These constituent systems are separately discovered, selected and composed at run-time or design-time to form a more complex system to fulfill a specific mission. It is an integration of autonomous systems that are geographically distributed and support continuous evolution. These systems are functionally and managerially independent. These systems on integration, share their resources and services to serve a larger, complex and unique functionality that is not possible to achieve otherwise.

SoS is a larger system that performs a function not performable by one of the constituent systems alone, thus it creates emergent behavior. Constituent systems fulfill their own objectives. If they are disassembled from the encompassing SoS they continue to operate to fulfill their own objectives and tasks. They are managed for their own objectives rather than the objectives of the whole SoS. Intrinsic characteristics of SoS are: *(1) Operational independence of systems:* If the SoS is disassembled into its component systems these systems must be able to usefully operate independently; *(2) Managerial independence of systems:* The component systems are separately acquired and integrated but maintain a continuing operational existence independent of the SoS; *(3) Geographical distribution of systems: (4) Evolutionary development of SoS: (5) Emergent behavior of SoS:* In addition, characteristics of Open-World SoS are the *unpredictable environment* and *unpredictable constituents*.

ISO/IEC/IEEE 42010 International Standard [1] defines a software-intensive system as any system in which software influences the *design*, *implementation*, *deployment*, and *evolution* of the system as a whole to encompass individual applications, subsystems, systems-of-systems, product lines, product families, whole enterprises and other aggregations of interest.

Self-adaptation enables a software system to reason about itself and adapt autonomously to achieve particular quality objectives in the face of uncertainties and change. Central to the realization of self-adaptation are feedback loops that monitor and adapt managed parts of a system when needed. Studies conducted in the field of self-adaptation have primarily focused on centralized and hierarchical control in self-adaptation, which is not applicable to systems that are inherently decentralized. Realizing self-adaptation in a SoS where no single entity has the knowledge and authority to supervise and adapt the constituent parts raises fundamental engineering challenges. [2]

In Software-intensive SoS, software essentially influences the analysis, design, architecture, implementation, deployment, and evolution of the system in itself. Software is essential to enable the behavior of these systems. It encompasses single systems and aggregations of interest, i.e. Systems-of-systems. Software-intensive system-of-systems constituents are themselves systems

B. Formal methods

Formal methods have a mathematical foundation. They provide a formal foundation in requirement specification, architecture definition, implementation, testing, maintenance and evolution of large-scale software. In industrial projects, multiple levels of formal methods are applied at different stages of software development life cycle depending upon the degree of criticality of the software project. The major emphasis is on the application of formal methods at the earlier stage of software development life cycle (i.e. specification and design).

At software design level formal methods are used to refine data using state machines, abstraction functions and simulation proofs while at implementation level code verification may be done by *theorem proving* or *inductive assertions*. The major emphasis will be the application of formal methods at the earlier stage of software specification and design. A number of researchers have conducted research surveys for the industrial use of formal methods. Formal methods are useful in the development and certification of critical systems. [12]

C. Correctness properties

Correctness properties play important role in system verification. Correctness properties of safety and liveness complement each other. Safety alone or liveness alone is not sufficient to ensure system correctness. The safety property is an invariant which asserts that "*something bad never happen*", that an acceptable state of affairs is maintained. For example consider a power reactor generating electricity; the reactor temperature should never exceed 100 degrees Centigrade to assure safe and efficient working. The property which assures that a power reactor temperature would never exceed 100 degrees Centigrade is a safety property.

[13] have defined safety property *S = {a1, a2 … an}* as "*a deterministic process that asserts that any trace including actions in the alphabet of S, is accepted by S*". ERROR conditions are like exceptions which state what is not required. In complex systems, safety properties are specified by directly stating what is required.

The liveness property asserts that "*something good happens*". It describes the states of a system that an agent must bring about given certain conditions. One of the most significant methods to ensure correctness of large-scale system is to use formal methods.

II. Motivation

The functions performed by a critical system have become considerably software-intensive. A critical system has to satisfy specific quality attributes like *liveness* and *safety*. As a critical system has to operate in open environment in which it interact and elaborate with other systems, satisfaction of the requirements through traditional offline top-down engineering no longer suffice. Guaranteeing correctness by construction is not possible for large-scale critical systems in which boundaries are no longer fixed and services provided by other systems will come and go in unpredictable ways. Such systems need to tolerate faults in the face of change; in short they need to be *resilient*. Building and managing resilient large-scale critical systems call for a fundamental shift in engineering vision in which satisfaction of requirements has to be realized via online collaboration among autonomous components.

III. Material and Methods

A. Objectives

The major objective of resilient architecture for software intensive system-of-systems is to develop efficient and robust approach for building resilient large-scale critical system-of-systems. Resilient Architecture is centered on four pillars:

1) Self-adaptation as a technique to achieve resilience: As large-scale critical system-of-systems are long-lived systems, they have to be prepared for openness. In an open environment, the context of the system can change at any time, availability of resources may change, services may evolve, services may disappear or new services may become available. To enable a system to deal with these dynamics it must be self-adaptive. A self-adaptive system is goal-oriented, it is aware of its context and reasons upon it, it coordinates with other systems in its environment and adapts itself with changing operating conditions.

2) Executable language for architecture: Designing and realizing self-adaptive large-scale critical systems requires suitable models at appropriate levels of abstraction. This calls for an innovative description and executable language for architecture which seamlessly integrates multi-view modeling with runtime model evolution. Support for multi-view modeling is crucial for two reasons. On the one hand, it enables the specification of different perspectives on the system and its environment according to the interests of the variety of system stakeholders. On the other hand, it enables the specification of the appropriate models of critical systems that are needed for automatic adaptation. Support for runtime model evolution is crucial to enable model adaptation to changes in the environment, possibly in unpredicted ways.

3) Formal foundation: Since large-scale critical systems have a number of mandatory requirements, a sound formal foundation is a prerequisite for the engineering approach. Assuring the qualities requires a rigorous specification of semantics of models and a formal understanding to enable automatic verification of model adaptations. For example, guaranteeing a safe adaptation in a decentralized system requires safety along the path of subsequent local adaptations, which demands formally founded methods and techniques.

4) Runtime execution platform: Automatic adaptation and evolution of critical systems requires a suitable runtime execution platform. Key aspects of this platform are automatic

and decentralized discovery of components services in large-scale open systems, goal-oriented decision making and coordination for adaptation, verification of fragments of the specification assuring the required qualities under adaptation, and automatic execution of system adaptation, based on the connection of the runtime models with the underlying implementation. A key quality aspect of the platform is scalability. In particular, the platform should support the realization of the applications' quality of service requirements in large-scale distributed settings.

B. Scope of Study

1) Security is not considered as a primary concern for the analysis and design of the resilient architecture for critical Software-intensive System-of-Systems (SiSoS).

2) Semantic interoperability (i.e. information exchange among systems) should be unambiguously defined.

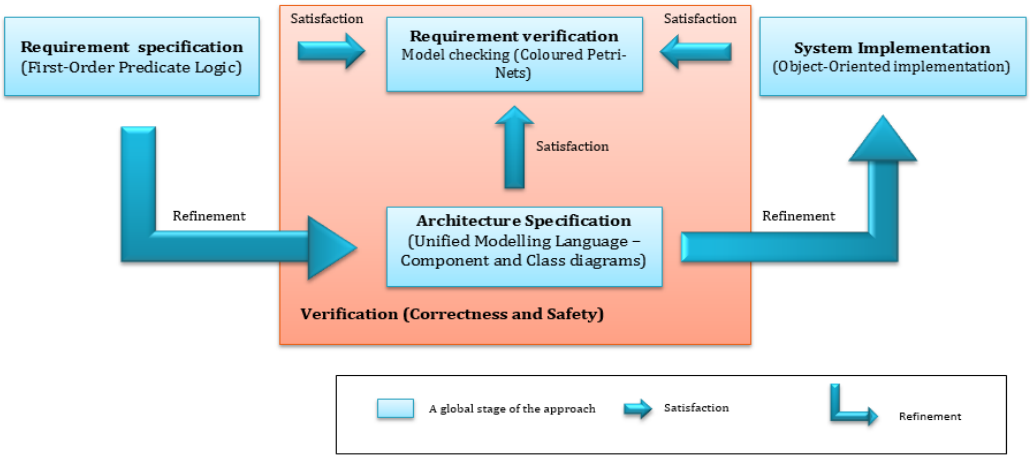

Fig. 2. The four phases of resilient architecture for critical Software-intensive System-of-Systems (SiSoS)

3) The degree of automation depends on the context from human designed to human-in-the-loop to fully automatic.

4) The scope of automatic verification depends on the support present in the formal method or language chosen.

C. Formal verification and Architecture definition

The formal foundation for specifying and verifying behavior-oriented software architecture developed in the European ArchWare [11] project offers a sound basis upon which the envisioned next generation description and executable language for architecture can be built. In addition, the lessons learned in ArchWare [11] with defining and designing architectural languages and supporting tools are highly valuable for Resilient Architectures.

The proposed approach has four major phases of requirement specification, requirement verification, architecture specification, and system implementation.

The architecture of an Information Management System has been specified, verified, and designed to validate the approach. The first phase is of requirement specification. The requirements are specified in *First-Order Predicate Logic*. There is a satisfaction relation between requirement specification and requirement verification.

Requirements are modeled and verified by Coloured Petri-Nets. The structural architecture is specified by using UML Component and Class diagrams, the behavioral architecture is modeled and verified by Coloured-Petri Nets. This architecture is refined into object-oriented implementation.

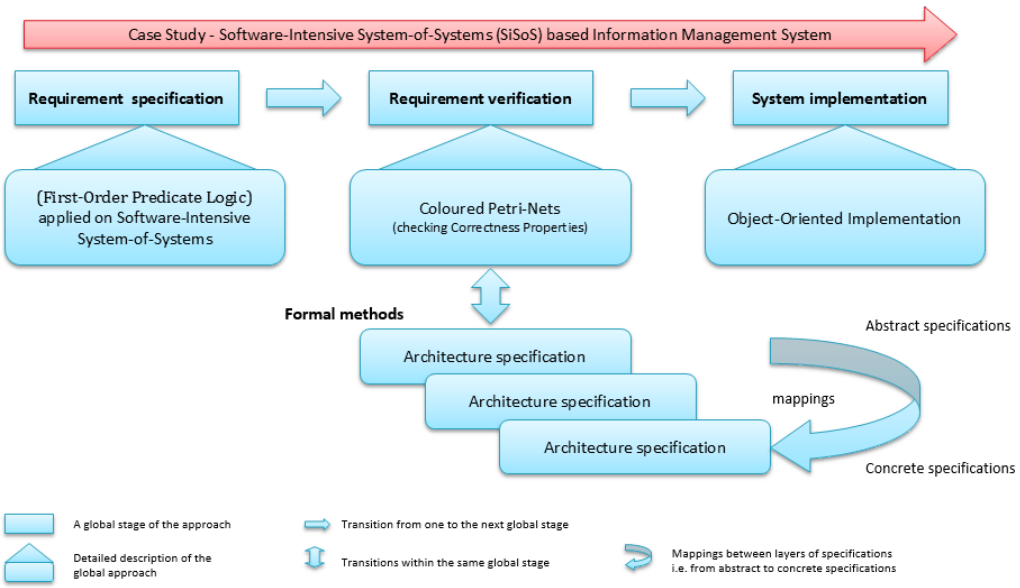

Fig. 3. The four phases of resilient architecture for critical Software-intensive System-of-Systems (SiSoS)

There is a satisfaction relation between system implementation and requirement verification. The system is implemented by using object-oriented implementation. The implementation satisfies the verification specifications.

IV. Results and Discussion

The expertise acquired during the proposition of decentralized control for autonomous robotic transport agents [14][15] proved to be valuable for the design of resilient architecture for SiSoS. In this research project, multi-agent system architecture is formally verified and developed to endow an automatic transportation system with advanced self-managing capabilities. Although this control system has fixed boundaries, the knowledge and expertise acquired from decentralizing control in this complex domain provides a substantial basis upon which resilient architecture can be built on.

As a result of this work, a resilient architecture for SiSoS has been proposed. The proposed approach is centered on formal verification of correctness properties. This approach is based on a combination of formal methods and techniques. By following this approach a resilient architecture for Software-intensive System-of-Systems can be specified, formally verified and implemented. As our previous expertise and experience is on decentralized control of autonomous robotic transport agents [14][15], our future goal is to use resilient architecture for Software-intensive System-of-Systems.

References

[1] ISO/IEC/IEEE 42010:2011(E), "ISO/IEC/IEEE International Standard for Systems and Software Engineering – Architectural description," 2011.

[2] Danny Weyns and Jesper Andersson. "On the challenges of self-adaptation in systems of systems". In *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems* (SESoS '13), Paris Avgeriou, 2013, Carlos E. Cuesta, José Carlos Maldonado, Elisa Y. Nakagawa, Khalil Drira, and Andrea Zisman (Eds.). ACM, New York, NY, USA, 47-51, 2013. DOI=http://dx.doi.org/10.1145/2489850.2489860

[3] Office of the Deputy Under Secretary of Defense for Acquisition and Technology, *Systems and Software Engineering, System engineering guide for systems of systems*, ODUSD (A&T), 2008, version 1.0.

[4] M. W. Maier, "Architecting principles for systems-of-systems," *Systems Engineering*, vol. 1, no. 4, pp. 267–284, 1998.

[5] D. Firesmith, "Profiling systems using the defining characteristics of systems of systems (SoS)," *Software Engineering Institute*, Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. TN-001, 2010.

[6] A. P. Sage and C. D. Cuppan. "On the systems engineering and management of systems of systems andfederations of systems". *Information Knowledge Systems Management*, 2(4):325-345, Dec. 2001.

[7] J. Boardman and B. Sauser. "System of systems-the meaning of of". In *International Conference on Systemof Systems Engineering*. IEEE, 2006.

[8] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longsta_, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau. "Ultra-Large-Scale Systems - the software challenge of the future". *Technical report*, SEI, Carnegie Mellon, 2006.

[9] B. Blanchard and W. Fabrycky, *Systems Engineering and Analysis*, 3rd Edition. Prentice Hall, pp. 2, 1998.

[10] D. A. DeLaurentis and W. A. Crossley, "A taxonomy-based perspective for systems of systems design methods," *In Proceedings of the 2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 1. USA: IEEE, 2005, pp. 86–91.

[11] F. Oquendo, B. Warboys, R. Morrison, R. Dindeleux, F. Gallo, H. Garavel, C. Occhipinti, "ArchWare: Architecting Evolvable Software," *Proc. 1st European Workshop Software Architecture* (EWSA 2004), LNCS 3047, Springer-Verlag, 2004, pp. 257-277.

[12] J. Woodcock, P. G. Larsen, J. Bicarregui and J. Fitzgerald, "Formal Methods and Experience", *ACM Computing Surveys*, Vol 16, No. 4, Article 19, October 2009.

[13] J. Magee, and J. Kramer, *Concurrency: State Models and Java Programs*. John Wiley and Sons, 2nd edition, 2006.

[14] N. Akhtar, "Contribution to the formal specification and verification of multi-agent robotic systems". *PhD thesis*, Ecole Doctorale, Laboratory VALORIA, University of South Brittany, 2010.

[15] N. Akhtar, Y. L. Guyadec, and F. Oquendo, "FORMAL SPECIFICATION AND VERIFICATION OF MULTI-AGENT ROBOTICS SOFTWARE SYSTEMS: A Case Study". *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART 09)*. Porto, Portugal, January 19-21. INSTICC Press, 2009.