Editorial

System and software architectures

Just about every system has an architecture — be it a computer system, a software system, a social system, a manufacturing system, or even a human body. The architecture of a system essentially is about the parts of the system and the connections between the parts. For example, the parts of a computer (system) include a CPU, a main memory, a disk drive and several input and output devices. These parts are then connected through various serial or parallel ports, cables, memory bus, etc. Taking another example, the parts of a university as a social system include students, professors, courses, classrooms and administrators. These parts are then connected with one another through such inter-relationships as students-in-a-course and instructors-of-a-course.

There seems to be a reason, of course, for the existence of each and every architecture. The parts of each architecture seem to interact with one another in order to (help) achieve certain goals, which may be only implicit or explicitly stated as such. For example, the parts of a computer system may interact with each other to accept commands from the users of the system, execute the commands and produce the desirable outputs. Similarly, the parts of a university may socially interact with each other, perhaps through various workflows, to provide high quality education, produce good research results and serve the community.

The type of system of particular concern in this special issue is software systems. A software system is different from other system types in many ways. One important difference lies in the notion of an intended application domain — which itself is a system. A software system is supposed to help the users of the intended application domain system achieve their goals. For this to happen, it would be desirable for the goals of a software system to align with the goals of the users of the intended application domain system. The various services offered by a software system should ultimately help the users of the intended application domain system achieve their goals, by helping them carry out the various tasks individually and also the various interactions with one another: help on the parts and connections among the parts — i.e., help on the architecture of the intended application domain system. In a nutshell, the architecture of a software system should be designed to meet the goals of a software system, which in turn should provide help on sustaining the architecture of the intended application domain system.

Building a high quality (software) system, of course, is a critical goal for software developers. Consider the effect of poor quality software — human lives are lost and a huge
sum of money has been wasted with embarrassments – for example, the Ariane rocket blow-up on lift-off, the Mars Climate Orbiter crash, the failure of ATC systems, viruses, spyware, intrusions.

As the blueprint for the system and software, the architecture of the system is an important factor in determining the quality of the final system (including software). Most of the qualities of the final system, such as performance, security, adaptability and maintainability, are decided during the architectural phase of the system (or software development). But how does one go about developing architectures?

In order to answer this question, the 3rd International Workshop on System/Software Architectures (IWSSA’04) was organized in June, 2004, in Las Vegas, as part of the International Conference on Software Engineering Research and Practice, 2004. IWSSA’04 was a huge success, judging from the quality of papers that it attracted, the relevance of the papers to the workshop topic and the audience participation. Selected papers from among those presented at IWSSA’04 were chosen for inclusion in this special issue — the selection was based on audience interest and feedback from the reviewers. All the selected papers were extended and revised for this special issue. In our opinion these papers give a glimpse into the fantastic world of system and software architectures for various domains.

The first paper, entitled “BITAM: An engineering-principled method for managing misalignments between business and IT architectures”, by Hong-Mei Chen, Rick Kazman and Aditya Garg, tackles the important problem of aligning business and IT architectures. Generally, the better aligned these architectures are, the more advantageous it is for the business organization. However, mismatches between the architectures can easily arise since changes in business rules and strategies usually occur at a much faster rate than can be accommodated by IT. This paper presents BITAM (Business IT Alignment Method), which gives a series of twelve steps for managing and reducing the misalignment between the business and IT architectures. BITAM defines mappings between three layers of a business system: business models, business architectures and IT architectures; misalignments are treated as improper mappings between these areas. BITAM is validated by applying it to a specific system of a Fortune 100 company.

The second paper, entitled “Self-healing component in robust software architecture for concurrent and distributed systems”, by Michael E. Shin, develops an approach to designing self-healing components for concurrent and distributed systems. A self-healing component can monitor itself and heal itself in the case of problems. To allow this to happen, each component has two layers — the services layer and the healing layer, and each layer is designed independently. The services layer provides the component functionality, while the healing layer provides the monitoring and healing capability for the component. The healing layer uses a systematic process to ensure continued proper operation of the component before and after any “sickness”. The approach is illustrated by using an elevator system consisting of three high level components.

The third paper, entitled “Evaluating the performance of architectures in MASCOT”, by Pere P. Sancho, Carlos Juiz and Ramon Puigjaner, describes the prototype, MASCOTime, that helps to evaluate the performance of architectures developed using MASCOT (Modular Approach to Software Construction Operation and Test). Using MASCOTime, architects can select those architectures, if any, that meet the minimum performance
requirements for the system. MASCOTime is a discrete-event simulator for MASCOT designs that mainly describes the service times of the functional components and analyzes them. The MASCOTime code is embedded into MASCOT Description Language for a component and a translator converts the embedded code into a Java program that gets linked with the MASCOTime library for simulation. The practicality of MASCOTime is illustrated by applying it to a radar management system.

The fourth paper, entitled “A calculus for concurrent system with higher-order streaming communication”, by Masaki Murakami, is interesting because it explores the possibility of applying formal concepts to a concurrent and distributed system such as the Internet. One of the biggest advantages in using the Internet is the ability to send and receive programs during computation, for example, in executing Java applets and linking to different pages using (downloaded and executed) HTML code. This ability can be captured by the notion of “mobility” of program codes and can be formally represented using higher order terms. Thus sending and receiving programs can be modeled as sending and receiving higher order terms and modified higher order π-calculus is used to manipulate these terms. The modification involves adding a new operation called input streaming and equivalence relations are established using the idea of barbed bisimulation.

The fifth paper, entitled “A Product Line engineering practices model”, by François Coallier and Roger Champagne, proposes a model for the successful implementation of a software product line. The software product line refers to the practice in industry of reusing various software artifacts in newer products in order to remain competitive. The model proposed draws ideas from SEI model and VRAPS model but focuses on the system level rather than the software level. The proposed model suggests 31 practice areas that have been grouped into five categories based on activities required for the categories. One of the objectives of the proposed model is to make it easy to integrate with existing development methods in industry. The model has been used for an industry sponsor and represents work in progress.

The sixth paper, entitled “Performance modeling and analysis of software architectures: An aspect-oriented UML based approach”, by Kendra Cooper, Lirong Dai and Yi Deng, establishes a framework that uses an aspect-oriented approach to model and analyze performance characteristics of distributed real-time systems. This framework automatically translates extended UML designs into a formal notation – the architecture description language, Armani – and using the existing tool support for the formal notation, analyzes the designs. The performance aspects analyzed include response time and resource utilization for the designs and helps detect components in the system that do not meet the performance criteria. This approach is illustrated by applying it to the Domain Name System.

The seventh and the last paper, entitled “Supporting ATAM with a collaborative Web-based software architecture evaluation tool”, by Piyush Maheshwari and Albert Teoh, lists the social and environmental problems in application of SEI’s Architecture Tradeoff Analysis Method (ATAM) by global commercial companies. Important issues and potential weaknesses in applying ATAM without suitable modifications have been discussed. One of the ways to overcome some of the problems is to use the Internet Advantageously — an Internet-based tool called the ATAM Collaborative Environment has been developed which will provide a more uniform interface for all stakeholders in software development.
to interact through. In this way, social and cultural problems could be mitigated for the global companies.

We hope the readers enjoy the viewpoints presented by the papers in this special issue on the important topic of system and software architectures. We would like to thank all the participants of IWSSA’04 for their wonderful cooperation. In particular, we would like to thank the reviewers of the papers for their thorough and excellent feedback: Stan Jarzabek, National University of Singapore, Singapore; Rick Kazman, Software Engineering Institute, USA; Juan Ramil, Open University, UK; Murakami Masaki, Okayama University, Japan; Carlos Juiz, University of the Balearic Islands, Spain; Philippe Aniorte, LIUPPA IUT de Bayonne, France; Vespe Savikko, VTT, Finland; Kendra Cooper, The University of Texas at Dallas, USA; and Yeong-Tae Song, Towson University, USA. We would like to thank Dr. Hamid Arabnia, the organizer of the multiconference at Las Vegas, for the support provided to us. And last but not least, we would like to thank Dr. Jan Bergstra, Journal of Science of Computer Programming, for agreeing to release this special issue, and Dr. Bas van Vlijmen, University of Amsterdam, for providing encouraging support throughout the preparation of this special issue.

Lawrence Chung*
Department of Computer Science,
University of Texas at Dallas,
Richardson, TX 75083,
United States
E-mail address: chung@utdallas.edu.

Nary Subramanian
Department of Engineering,
Hofstra University,
Hempstead, NY 11549,
United States

Available online 3 March 2005

* Corresponding editor.