International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:6, No:7, 2012

Aspect Oriented Software Architecture

Pradip Peter Dey, Ronald F. Gonzales, Gordon W. Romney, Mohammad Amin, Bhaskar Raj Sinha

***Abstract*—**Natural language processing systems pose a unique challenge for software architectural design as system complexity has increased continually and systems cannot be easily constructed from loosely coupled modules. Lexical, syntactic, semantic, and pragmatic aspects of linguistic information are tightly coupled in a manner that requires separation of concerns in a special way in design, implementation and maintenance. An aspect oriented software architecture is proposed in this paper after critically reviewing relevant architectural issues. For the purpose of this paper, the syntactic aspect is characterized by an augmented context-free grammar. The semantic aspect is composed of multiple perspectives including denotational, operational, axiomatic and case frame approaches. Case frame semantics matured in India from deep thematic analysis. It is argued that lexical, syntactic, semantic and pragmatic aspects work together in a mutually dependent way and their synergy is best represented in the aspect oriented approach. The software architecture is presented with an augmented Unified Modeling Language.

***Keywords*—**Language engineering, parsing, software design, user experience.

I. Introduction

UNTIL recently, Object Oriented Design (OOD) was considered as one of the best approaches for designing complex software systems [1]-[4]. Recent investigations into separation of concerns have led to the considerations of some new approaches including Aspect Oriented Design (AOD) [5]-[8]. This paper examines important software design issues and presents justifications for AOD with a case study from natural language processing. Architectural design, detailed design and design reviews provide the most important steps in a cost effective software development process. Software engineering activities are goal directed in order to produce working software in a timely manner within some cost constraints. For any complex computer based system, software architecture plays a very important role in its success or failure. According to Pressman [1: page 223] “One goal of software design is to derive an architectural rendering of a system”. Multiple representations of software architecture are recommended for providing different views of a complex system in order to clarify the structure of the system, which comprises software components and the externally visible properties of those components. Software architecture is “the overall structure of the software and the ways in which that structure provides conceptual integrity for a system” [3]. It is also known as high level design since conceptual integrity is clarified at a high level of abstraction.

Pradip Peter Dey, Ronald F. Gonzales, Gordon W. Romney, Mohammad Amin and Bhaskar Raj Sinha are with National University, 3678 Aero Court, San Diego, CA 92123, USA. They are now with the School of Engineering, Technology and Media (phone: 858-309-3412; fax: 858-309-3420; e-mail: pdey@nu.edu; rgonzales@nu.edu; gromney@nu.edu; mamin@nu.edu; bsinha@nu.edu)

According to Braude and Bernstein [4: page 438], "A software architecture describes the overall components of an application and how they relate to each other." The emphasis on components were considered very productive in OOD, although this is recently questioned for systems with crosscutting aspects. Security aspects are often considered to be spread over multiple components in a complex manner that defies most variants of OOD approaches. The best architectural practices are rarely published and often inferred from excellent products [9]. In practice, software architectural design is immensely challenging, vastly multifaceted, strikingly domain based, perpetually changing, rarely cost-effective, and deceptively ambiguous. Multiple representations and intuitive explanations are often provided in order to lessen the difficulty of interpretations of software architecture.

II. Background

Practitioners and theoreticians have been debating about software development approaches for a long time. Opposing views are often presented with effective metaphors. Donald Knuth initially [11] suggested that software writing is an art. David Gries [12] argued it to be a science. Watts Humphrey [13] viewed it as a process. In recent years, practitioners have come to realize that software is engineered [1]-[2], [4], [14]-[17]. The scientific foundation of software engineering is not fully understood. That is, we do not understand it the way we understand chemistry as the scientific foundation of chemical engineering. Software architectural design is based partly on computer science and partly on behavioral sciences and intuitive judgments although there were some minor attempts to establish “software science” [2] as the primary basis for software architecture.

It is often suggested that software architectural design is creatively built from requirements analysis in an iterative process [1], [4], [13]-[19]. In this process, after some initial requirements analysis a software architectural representation is developed and then the requirements analysis is augmented on the basis of a combination of software architecture, new or changed requirements or some other factors which in turn leads to a revised software architecture. The architectural representation developed in this manner traditionally consisted of components and their relationships with the primary assumption that the software is composed of these components. The components were obtained mainly by separating and grouping concerns or related computational elements. Recent studies suggest that certain concerns cannot be easily localized and specified with individual architectural units such as components [5]-[8]. These crosscutting concerns are best represented as architectural aspects in an architectural design. In order to highlight architectural aspects of AOD, we will consider an interesting case presented below.

III. Language Processing

Aspects of natural language processing are stimulating for many reasons, especially for the intricate relationship among lexical, syntactic, semantic and pragmatic facets. "We speak informally of the sound and meaning of a word, the way it is pronounced, and what it means" [20: p170]. It is generally accepted among experts that the meaning of a sentence is composed from the meaning of its words. An analysis of language for producing a meaningful interpretation is the most crucial part of a of natural language processing system. However, such an analysis is one of the most challenging problems in computer science [21]-[28]. Understanding the nature of challenge requires a thorough study of all major aspects of natural language and their proper relationships. Although substantial progress has been made in lexical processing, controversies on syntactic, semantic and pragmatic aspects remain unresolved. Each of the processing aspects mentioned above is easy to understand, but difficult to formalize for efficient processing. Structure and interpretation of natural language have been among the most elusive problems in formal modeling. Given an input sentence such as "The frog jumped", computational problems can be explained as follows. Lexical processing of the sentence is performed by searching the English lexicon for each of the words and recognizing the word "The" as a determiner, "frog" as a noun and "jumped" as a verb [23]-[24]. Syntactic processing is carried out by finding relationships among the words in the sentence, building constituents and providing a structural description [22]-[24], called a parse tree or sentence diagram, such as the one shown in Figure 1. It is usually assumed that lexical analysis precedes syntactic processing, although their interdependent nature is also recognized.

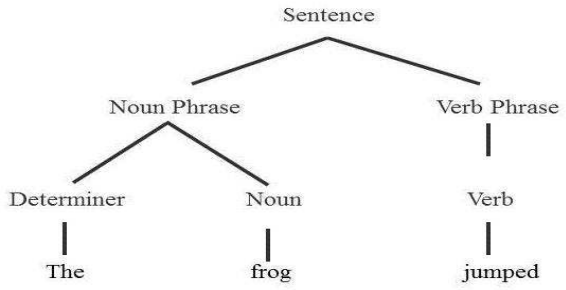

Fig. 1 A Parse Tree for "The frog jumped"

The semantic aspects are processed by analyzing the sentence into an interpretation that can be utilized for reasoning, knowledge representation, database updates, information retrieval or other uses. One of the ways to represent meaning is through first order logic or first order predicate calculus, although semantic networks, case-frames, modal logic and other forms are also popular [23], [25], [28]. In first order logic, the meaning of "The frog jumped" can be represented as: $(\exists x)$ [Frog(x) ^ Jump-Past(x)]. Other representations of meaning will be discussed in section-3 of this paper.

Language use in linguistic and extra linguistic contexts is the focus of the pragmatic analysis where commonsense reasoning plays an important role [26], [28]. This paper addresses some of the central problems in these areas and presents a new software architecture in order to provide alternative analyses in semantics crucial for language understanding. Semantic processing is usually performed compositionally, that is, the meaning of larger linguistic units is usually derived by combining the meanings of smaller ones. However, the rules of composition and the nature of semantic representation are not yet fully understood posing a major challenge for the development of a fully functional computational linguistic system with incomplete knowledge. Currently, several alternative approaches seem to be promising, especially in the semantic area where progress has been limited. A software architecture that accommodates all major semantic approaches including denotational, operational, axiomatic and case frame methods is presented in this paper with justifications.

The software architecture presented in this paper follows the current practices in identifying the major components and their relationships. A significant aspect of the research is that components of a language processing system need to be best organized for computationally efficient and linguistically adequate language engineering. According to the general guidelines and best practices in software engineering [11-13], loosely coupled components are preferred over tightly coupled components in a software system.

The architectural design of a natural language processing system is one of the most difficult problems in computer science. It is immensely complicated, highly multifaceted, extravagantly contextual, deceptively ambiguous, and strikingly controversial. The underlying computation is proven to be an NP-Complete problem [14-15]. One of the challenges is the interaction among lexical, syntactic, semantic and pragmatic processing in the presence of ambiguity. In addition to computational complexity, there are additional problems of arrangements and relationships among the components where the aspects of processing take place. Experts often try to solve these problems in a principled way using a pipeline architecture where the output of lexical processing is input to the syntax analyzer, whose output is input to the semantic analyzer and so on. This architecture is schematically shown in Figure 2. This architecture is similar to the one used successfully in compilers [16-17]. An advantage of this architecture is that it intuitively resembles an assembly line and it separates concerns in different loosely coupled components. The elegant features of this architecture are made very clear by may practitioners [4, 16, 17]. However, natural languages are not fully specified formally and do not lend themselves to compiler techniques adequately. In the pipeline architecture, the interactions among the components are not flexible enough to handle some of the complex natural language problems such as syntax-semantics interactions.

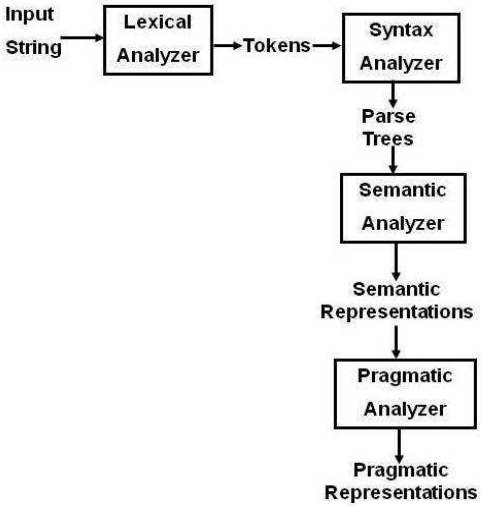

Fig. 2 A Pipeline Architecture for Natural Language Processing

The design and implementation of natural language require careful consideration of interactions among all components. After initial requirements analysis the software was designed using the Model-View-Controller architecture [18-19] and an initial prototype was developed following the iterative development process. After several iterations, it was realized that the separation between the View and Controller components did not have any advantages because the Controller needed to work closely with the view and access the View elements repeatedly. The View and Controller elements can be combined into a single component called User Interface. The Model is responsible for processing the domain information; it includes the lexical, syntactic, semantic and pragmatic components which seem to work together with aspects. This architecture is presented in Figure 3 which allows more robust interactions among its components. All major components are shown in Figure 3 using the Unified Modeling Language (UML) notations augmented with architectural aspects shown in shaded diamonds. In the augmented UML, the components are presented with required interfaces and provided interfaces. A required interface is shown with a small semicircle attached to a component. A provided interface is shown with a small circle attached to a component. The semantic processing needs to compose meaning of a sentence from its parts. The rules of composition are derived from syntax because the constituent structures of syntax are properly guided by these rules. In addition, the aspects of semantic analysis may include denotational, operational, axiomatic and case-frame semantics, because these approaches complement each other in order to provide a comprehensive treatment of meaning. The architecture in Figure 3 is, therefore, composed of UML based components augmented with Aspects Oriented (AO) features [7-8], [29]. A detailed justification of the architecture, presented in the next section may help in making a strong case for the architectural design.

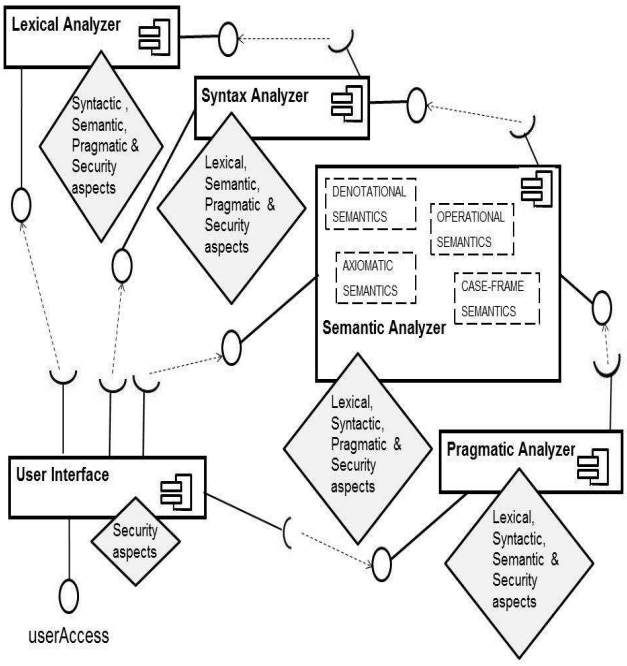

Fig. 3 An Aspect Oriented Software Architecture for Natural Language Processing

IV. Justifications

The justifications for software architecture come from different sources. One of the significant assumptions behind the AOD architecture is that the semantic and syntactic components need to work together. This assumption is supported by the fact that all major semantic approaches are compositional, but the rules of composition are provided by syntax [22]-[25]. Some studies also demonstrate that semantic information is required for syntactic decisions [30-31]. A verb like “pretend” is neither transitive nor intransitive but takes a sentential complement as shown in (1). From the unacceptability of (2) and (3) and similar examples, Green [30, p 10] concludes that semantic information is required in making syntactic predictions. That is, syntactic sub-categorization of verbs and imposition of selectional restrictions are not sufficient to solve these problems [30]-[33]. It is to be noted that unacceptability of strings is indicated by a preceding star, *****.

(1) John pretended that he was in Paris.
(2) * John pretended.
(3) * John pretended Paris.

It is not easy to decide how to combine syntactic and semantic information. To justify the AO software architecture for natural language processing, four interesting problems that require synergistic relationship among various components are considered below.

A. Ambiguity

A grammar is ambiguous if and only if it assigns two or more syntactic structures to at least one input string.

Loosely coupled modules of syntax and semantics do not adequately support ambiguity treatment without cross-cutting aspects. For example, a sentence like "Old men and women danced" admits two distinct semantic interpretations, each of which corresponds to a syntactic structure. The syntactic structure given in Figure 4, according to most grammars, including tree adjoining grammars [22], [33], supports the interpretation that the adjective "Old" modifies the entire conjoined noun phrase "men and women" meaning both men and women are old. On the other hand, the syntactic structure given in Figure 5 supports the interpretation that "Old" modifies "men" only, because "Old men" form a noun phrase constituent [$_{\text{Noun Phrase}}$ ($_{\text{Adjective}}$ Old) ($_{\text{Noun}}$ men)] whereas the noun "women" is not modified by "Old".

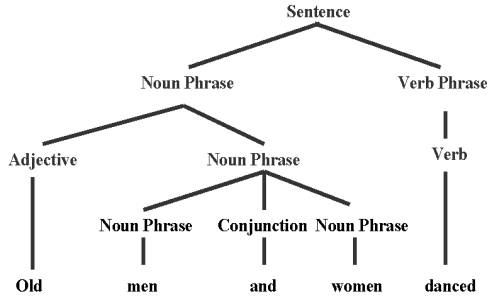

Fig. 4 A Parse Tree where an adjective modifies a conjoined Noun Phrase

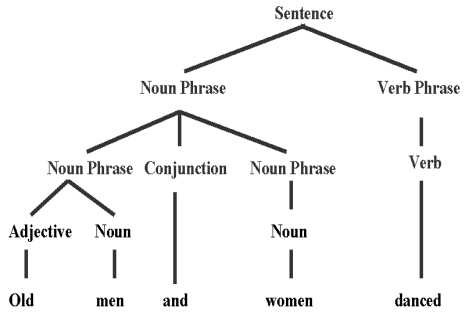

Fig. 6 A Parse Tree where an adjective modifies a Noun

Syntax and semantics together interpret this type of ambiguity better than semantics alone. The proposed AO architecture allows robust interactions among various components, including syntax and semantics. Often, lexical ambiguity gives rise to syntactic and semantic ambiguity. In the statement "Rice flies like sand" the word "flies" could be a noun or verb [23]. If "flies" is a noun and "like" is a verb then the interpretation would be "Rice flies are fond of sand". On the other hand, if "flies" is a verb and "like" is a preposition then the interpretation would be "Rice moves as sand moves".

B. Semantic Approaches

Semantics is one of the most challenging fields in language analysis and there is no clear winner among various competing semantic approaches. Therefore, it is reasonable to use all foremost semantic approaches for accommodating all key perspectives. The major approaches are explained as follows: (a) *Denotational Semantics*: This approach suggests that the meaning of a linguistic unit, such as a noun, is the entity it denotes. For example, in "Ernest killed himself" the person who got killed and the killer is denoted by the same individual. That is, "Earnest" and "himself" denote the same person. The statements "He killed herself" and "She killed themselves" are unacceptable because the subject and the object denotations are not identical and violate reflexive constructions. (b) *Operational Semantics*: This approach is also known as behavioral semantics and advocates that the meaning is best shown in the actions of a model, world, or virtual machine. Thus the meaning of the request "Please open the door" is best demonstrated by opening the door. The meaning of "delete" in a computer environment is the set of actions taken by the computer after the command is given. For every linguistic unit in a language, a Turing Machine can be built and executed on a universal Turing Machine [34] defining the operational semantics of that linguistic unit. (c) *Axiomatic Semantics*: According to this approach, the meaning of a linguistic unit is the set of consequences derivable from the linguistic unit in combination with a set of axioms. This is a proof theoretic approach utilizing mathematical logic, such as first order predicate calculus. Thus, the meaning of "Ernest killed himself" includes the consequences that "Ernest is not alive anymore," "Earnest is not drinking anymore," and so on. (d) *Case Frame Semantics*: Case frame semantics is popular with a number of practitioners including Fillmore [35-37]. Case frame semantics was originally developed in ancient India and was based on deep thematic relations among constituent parts of the sentence. Fillmore [35] pointed out that the noun phrase "the door" is the logical object of the verb "open" in all three sentences given in (4-6), but it is the syntactic subject in (4). Similarly the noun phrase "the key" is logically an instrument in both (5) and (6) but a syntactic subject in (5).

(4) The door opened.
(5) The key opened the door.
(6) The janitor opened the door with the key.

According to case-frame semantics, the underlying logical or thematic relations that need to be discovered and specified in semantic representation can be processed with deeper language analysis with robust interactions of all the components. Every instrument case that appears as a prepositional phrase (with-phrase) cannot be used as a subject of the same verb, resulting in (8) and (10) being unacceptable.

(7) The janitor ate spaghetti with the fork.
(8) * The fork ate spaghetti.
(9) The janitor ate spaghetti with eggs.
(10) * Eggs ate spaghetti.

A. Conjunctions and Disjunctions

Conjunctions and disjunctions are easy to understand intuitively. However, their meanings are difficult to specify without robust interactions of lexical, syntactic, semantic and pragmatic components. Logically, the order of conjuncts should not be a problem for meaning. However, the string in (12) is unacceptable for interpretation.

(11) She took poison and died.
(12) * She died and took poison.

In above examples, each conjunct is true separately. However, (11) is acceptable because the second conjunct is taken to be a consequence of the first conjunct. Similarly, a disjunction like "Don't move or I will shoot" requires special treatments. These types of conjunctions and disjunctions cannot be easily processed without the common sense reasoning of the pragmatic component, in addition to lexical, syntactic and semantic components, working together. Pragmatic information about speakers, hearers and audience is often needed for understanding consequences of natural language strings. Hearers of different cultural backgrounds have different interpretations when they are told that an immortal would die for his wife. These differences can be accounted for in a properly defined AO architecture.

V. Conclusion

The traditional view of loosely coupled independent components is not productive for designing natural language processing systems. The high level design for a natural language processing system with an AO architecture presented in this paper supports the synergistic relationship among lexical, syntactic, semantic and pragmatic components of the system. Without the architectural properties presented here, a language processing system is unlikely to process linguistic information adequately. The significance of this architecture is the synergistic relationship among lexical, syntactic, semantic and pragmatic components of the system. Future studies include a comprehensive language processing system implementation using this AO architecture along with a tree adjoining grammar and detailed low level design specifications in an iterative development process. In addition, evaluation of AO software along line suggested in [38]-[39] would be of general interest in this area.

Acknowledgment

The authors gratefully acknowledge the help and/or encouragements received from John Cicero, Hassan Badkoobehi, Byunggu Yu, Arun Datta, Jodi Reeves and many others during the preparation of this paper.

References

[1] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. (7th ed.), McGraw-Hill, 2010.
[2] Y. Wang, *Software Engineering Foundations*: *A Software Science Perspective*, Auerbach Publications, 2008.
[3] M. Shaw, and D. Garlan, "Formulations and Formalisms in Software Architectures", *Computer Science Today: Recent Trends and Developments*, Springer-Verlag LNCS, 1000, 307-323, 1995.
[4] E. Braude, and M. Bernstein, *Software Engineering: Modern Approaches*, (2nd Edition), John Wiley & Sons, 2011.
[5] C. Chavez, A. Garcia, U. Kulesza, C. Sant'Anna, C. Lucena. Taming Heterogeneous Aspects with Crosscutting Interfaces. *Journal of the Brazilian Computer Society*, 2006.
[6] E. Baniassad, P. Clements, J. Araujo, A. Moreira, A. Rashid, and B. Tekinerdogan, Discovering Early Aspects, *IEEE Software*, 2006.
[7] I. Krechetov, B. Tekinerdogan, and A. Garcia. Towards an integrated aspect-oriented modeling approach for software architecture design. In . In 8th Aspect-Oriented Modeling Workshop, Aspect-Oriented Software Development (AOSD) 2006.
[8] A. Navasa , M. A. Pérez , J. M. Murillo , J. Hernández. Aspect Oriented Software Architecture: A Structural Perspective, Proceedings of the Aspect-Oriented Software Development (AOSD), 2002.
[9] J. Hong, "Why is Great Design so Hard?", *Communications of the ACM*, July 2010.
[10] J. L. Azevedo, B. Cunha, and L. Almeida, "Hierarchical Distributed Architectures for Autonomous Mobile Robots: A case study", in *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, 2007.
[11] D. E. Knuth, *Seminumerical Algorithms: The Art of Computer Programming 2*. Addison-Wesley, Reading, Mass., 1969
[12] D. Gries, *The Science of Programming*. Springer, 1981.
[13] W. Humphrey, *Managing the Software Process*, Reading, MA. Addison-Wesley.
[14] I. Sommerville, *Software Engineering*, 9th Edition, Addison Wesley, 2010.
[15] S. Pfleeger, and J. Atlee, *Software Engineering*, Prentice-Hall, 2010.
[16] B. Agarwal, S. Tayal and M. Gupta, *Software Engineering and Testing*, Jones and Bartlet, 2010.
[17] F. Tsui, and O. Karam, *Essentials of Software Engineering*, 2nd Ed., Jones and Bartlet, 2011.
[18] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd Edition Addison-Wesley, 2003.
[19] J. Miller, and J. Mujerki, Editors, MDA Guide, Version 1, OMG Technical Report. Document OMG/200-05-01, http://www.omg.com/mda, 2003.
[20] N. Chomsky, *New Horizons in the Study of Language and Mind*, Cambridge University Press., 2000.
[21] R. Hausser, *Foundations of Computational Linguistics: Human-computer Communication in Natural Language*. (2nd ed.), Springer, New York, 2001.
[22] A. Abeille, and O. Rambow, *Tree Adjoining Grammars*. Univ. of Chicago Press., 2001.
[23] J. Allen, *Natural Language Understanding*. 2nd ed. Addison-Wesley, New York, 1995.
[24] P. Culicover, *Natural Language Syntax*, Oxford University Press., 2008.
[25] H. Alshawi, *The Core Language Engine*. MA: MIT Press., 1992.
[26] L. Iwanska, and S. Shapiro, (Eds.), *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*. AAAI Press. 2000.
[27] D. Jurafsky, *Speech and Language Processing: An Introduction to Natural Language Processing, computational linguistics and speech recognition*. Prentice Hall., 2000.
[28] A. Cruse, *Meaning in Language*: A*n introduction to Semantics and Pragmatics*. (2nd ed.). Oxford Univ. Press., 2004.
[29] R. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. (2nd Edition), Addison Wesley, 2005.
[30] G.M. Green, Semantics and Syntactic regularity. Fitzhenry & Whiteside Limited, Don Mills, Ontario, 1974.
[31] P. P. Dey, Y. Hayashi, and E. Battistella, (1989). A combination of strategies for parsing grammatical agreement in Hindi. *International Journal of Pattern Recognition and Artificial Intelligence*, 3, 1989, 261-273.
[32] R. D. Van Valin, *Exploring the Syntax-Semantics Interface*. Cambridge University Press, 2005.
[33] P. P. Dey, B. Bryant, and T. Takaoka, Lexical Ambiguity in Tree Adjoining Grammars, *Information Processing Letters*, 34, 1990, 65-69.
[34] D. Cohen, Introduction to Computer Theory , 2nd Edition, John Wiley & Sons, 1997.
[35] C. Fillmore, The case for case. In E. Bach & R. T. Harms (Eds.). Universals in Linguistic Theory. New York: Holt, Rinehart and Winston, 1968.
[36] C. Fillmore, Frames and the semantics of understanding. Quaderni di Semantica 6.2, 222-254, 1985.
[37] R. Schank, and R. P. Abelson, Scripts, Plans, Goals, and Understanding, Lawrence Erlbaum. 1977.
[38] B. Tekeinerdogan, and M. Aksit, "Classifying and Evaluating Architecture Design Methods", in M. Aksit (editor), *Software Architectures and Component Technology*, Kluwer Academic Publishers, 2002.
[39] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures*. Addison-Wesley, 2005.