

Granular Problem Solving and Software Engineering

Haibin Zhu, *Senior Member, IEEE*

Department of Computer Science and Mathematics, Nipissing University,
100 College Drive, North Bay, Ontario, P1B 8L7, Canada

Email: haibinz@nipissingu.ca

Abstract

Granulation is an important component of Granular Computing (GrC) as a problem solving paradigm. Specification and regulation of granulation are necessary in helping researchers and practitioners apply GrC into different applications. At present, there is insufficient investigation of this topic.

This paper defines concepts and mechanisms of problem solving, investigates the fundamental principles and processes of granulation, and demonstrates that the development of software engineering methodologies follows the basic idea of GrC: granulation. With the guide of granulation, this paper also proposes a new method of granulation by rolification (the act or the result of designing roles), i.e., roles and agents. The major contribution is establishing the foundations of granular problem solving and clarifying the relationships between granular problem solving and software engineering.

Keywords: problem solving, granular computing, granule, granulation, role, agent.

1. INTRODUCTION

Problem solving is an everyday activity in our lives. People learn to solve problems in early childhood. Every kind of job involves problem solving. Different jobs require various problem solving strategies, specialties and skills.

Granular Computing (GrC) is a general computing methodology using granules to establish an efficient computational model for complex applications [1] and believed to be a new type of problem solving paradigm [6, 7, 11-17]. Granules and granulation are no doubt the major components of GrC. However, there are few broad, formal and accurate definitions and analyses of granules and granulation. This paper proposes formal definitions for granules and granulations based on those of problems and problem solving. These definitions establish effective guidelines for the application of GrC.

Software Engineering (SE) has been in existence for more than three decades. It has led to the development of many concepts, structures, and methodologies for problem solving. It is now necessary to analyze the relationships between SE and Granular Problem Solving (GrPS). Roles and agents are new concepts and mechanisms in SE. They

can improve conventional SE structures and methodologies. Therefore, they provide a possible new way for granulation to be investigated.

The remainder of this paper is arranged as follows: Section 2 defines problems and problem solving; Section 3 analyzes the different forms and methods of granulation; Section 4 demonstrates the major SE methodologies in the sense of GrPS; Section 5 proposes a new granulation method based on roles and agents; Section 6 reviews the related work with GrPS; and Section 7 concludes the paper and discusses possible future work.

2. PROBLEM SOLVING

Problem solving involves many fundamental concepts and mechanisms such as induction, deduction, abstraction, classification, and decomposition. Abstraction is the first thinking methodology used when attempting to compose a new concept. Dictionaries [4] define a problem as “a question to be considered, solved, or answered; any question or matter involving doubt, uncertainty, or difficulty; a question proposed for solution or discussion; or a misgiving, objection, or complaint.” Whatever the problem, a real-world set of objects is involved. “A real-world problem normally consists of a web of interacting and interrelated parts [2]”.

Definition 1: Object. Everything in the world is an object. An object possesses identity and attributes (its **structure**).

The mark o denotes a specific object and O the set of all the objects.

Definition 2: Problem. A problem p is a special questionable object that needs to be solved. A problem can be defined as a tuple, $p ::= \langle v, a \rangle$, where, “ $::=$ ” expresses “is defined as”, v is an operation, and a is an object called parameter or argument, a may be null. It can be also expressed as $v(a)$.

In natural language, v is a verb and a is a noun. The mark p denotes a specific problem and \mathcal{P} the set of all the problems. Evidently, $\mathcal{P} \subset O$ is always true. For example, “do(homework)” is a problem.

Definition 3: Solution. A solution s is a way to solve a problem p or a concrete result object of the way. The mark s denotes a specific solution and \mathcal{S} the set of all the solutions. Evidently, $\mathcal{S} \subset O$ is always true. For example,

both “learn, understand, and do” and “the homework done” are solutions of problem “do(homework)”.

Definition 4: Problem solving. Problem solving for a problem $p \in \mathcal{P}$ can be defined as an activity to find a map $\langle p, s \rangle$ where $s \in \mathcal{S}$. The mark s denotes a specific solution and \mathcal{S} the set of all the problems.

In Definition 4, a problem p may be very simple or complicated. For a specific problem p , its map $\langle p, s \rangle$ may or may not exist.

Definition 5: Solvable problem. A problem p is solvable if there is one solution s making $\langle p, s \rangle$ exist.

The **properties** of problems and problem solving:

- 1) One problem may or may not have a solution.
- 2) One problem may have many (>1) solutions.
- 3) A solution may be concrete or abstract.
- 4) The set of solutions \mathcal{S} is expanding.

It means that a problem may have no solution for the time being but may have a solution in a later time.

- 5) There are many forms of solutions.

For example, the problem “landOnByACar (theMoon)” has no current solution; “the homework done” and “learn, understand and do” are solutions of the problem “do(homework)”, where, the former is a concrete solution but the latter an abstract one; the problem “callOnAPlane(aFriend)” had no solution in 1960s but has one now; mathematical problems have solutions in the forms, such as numbers, formula, equations, tables, and graphs; health problems have solutions in the forms, such as pills, surgeries, and transplants; and software problems have solutions in the form of software programs.

3. GRANULATION

Granules can be taken as the basic entities of knowledge [7]. “Granulation of a universe involves the decomposition of the universe into parts, or the grouping of individual elements into classes, based on available information and knowledge [18].” Granulation is a process of decomposition from whole to parts [9, 11-17].

Granulation in fact involves both composition and division. A granule is a group of objects that are drawn together based on some criteria [6, 7]. The definitions in Section 2 are actually a process of granulation. If we take granules as sets, granulation may split a set into smaller ones or consolidate others into a single set. The major aim is to make a problem easier to solve.

Definition 6: Granule. A granule p_g is a problem, a partial problem, or a combination of parts of the operation and parameter of a problem. A granule can be defined as $p_g ::= \langle v, a \rangle$, and the expression $v(a)$ denotes both a granule and a problem, i.e., a problem is also a granule (or **initial granule**).

A **final granule** is a problem that can be solved in a limited time by a person with relevant knowledge. A final

granule does not require splitting in order to be solved. The mark p_g denotes a specific granule, \mathcal{P}_g the set of all granules. Evidently, $\mathcal{P}_g \subset \mathcal{P}$ is always true.

Definition 7: Granulation. Granulation is an iterative process to find a set of final granules from an initial granule $v(a)$ by division or composition. The process stops when every granule is final.

From the viewpoint of division, a granule $v(a)$ can be split into three forms of granules, where “ \xrightarrow{d} ” is used to express “is divided into”: note that in the following discussions, m, n, m', n', i , and j are all integers.

- **Division 1:** $v(a) \xrightarrow{d} \{v_i(a) | m > 1, 0 \leq i < m\}$. It expresses that the operation v is split into m partial operations and each partial operation v_i can form a granule by acting on the parameter a .
- **Division 2:** $v(a) \xrightarrow{d} \{v(a_j) | n > 1, 0 \leq j < n\}$. It expresses that the parameter a is split into partial parameters a_j and the operation v forms a granule by acting on a partial parameter.
- **Division 3:** $v(a) \xrightarrow{d} \{v_i(a_j) | m > 1, n > 1, 0 \leq i < m, 0 \leq j < n\}$. It expresses that the operation v is split into partial operations v_i and the parameter a is split into partial parameters a_j and a partial operation v_i forms a granule by acting on a partial parameter a_j .

A complicated problem may be divided into many simple problems. A solution for one problem may be a part of another problem. For example, $(12+35) \times (34-23)$ (or $\times(+(12, 35), -(34, 23))$) is a problem. It can be divided into several simple problems, such as $12+35$, $34-23$, and 47×11 , where, 47 is the solution of $12+35$ and 11 is that of $34-23$.

The aim of granulation is to make a problem easy to understand and handle. Problem solving does not always require splitting. Sometimes, composition is required. From the viewpoint of composition, three forms are as follows corresponding to the forms of divisions respectively, where, “ \xleftarrow{c} ” is used to express “is composed from”:

- **Composition 1:** $\{(v^0, v^1, \dots, v^m)(a) | m > 1, 0 \leq m' < m\} \xleftarrow{c} \{v_i(a) | m > 1, 0 \leq i < m\}$. It expresses that a new operation (v^0, v^1, \dots, v^m) is composed from $m'+1$ partial operations of the m partial operations v_i and the new operation (v^0, v^1, \dots, v^m) can form a granule by acting on the parameter a .
- **Composition 2:** $\{v(a^0, a^1, \dots, a^n) | n > 1, 0 \leq n' < n\} \xleftarrow{c} \{v(a_j) | n > 1, 0 \leq j < n\}$. It expresses that a new parameter (a^0, a^1, \dots, a^n) is composed from $n'+1$ partial parameters of the n partial parameters a_j and the operation v can form a granule by acting on the new parameter (a^0, a^1, \dots, a^n) .
- **Composition 3:** $\{(v^0, v^1, \dots, v^m)(a^0, a^1, \dots, a^n) | m > 1, 0 \leq m' < m, n > 1, 0 \leq j < n\} \xleftarrow{c} \{v_i(a_j) | m > 1, n > 1, 0 \leq i < m,$

$0 \leq j < n$. It expresses that an new operation (v^0, v^1, \dots, v^m) is composed from $m'+1$ partial operations of the m partial operations v_i and a new parameter (a^0, a^1, \dots, a^n) is composed from the $n'+1$ partial parameters of the n partial parameters a_j and the new operation (v^0, v^1, \dots, v^m) forms a granule by acting on the new parameter (a^0, a^1, \dots, a^n) .

It seems easy to split or compose granules based on the different forms of granules. In fact, there are many difficulties to differentiate among many ambiguous and similar objects. The above discussion only concerns “what to do” but it is more complicated to implement “how to do”.

Granulation methods provide processes to obtain component granules from a whole granule or vice versa. For example, “build(car)” is a whole problem, it can be divided into partial granules, such as, “build(engine)”, “build(transmission)”, “build(body)”, “build(tire)” and “build(auxiliary)”. This process is **Division 2**. Granules “buy(software)”, “reuse(software)”, “makeByOwn(software)”, and “borrow(software)” can be composed into “build(software)”. It is **Composition 1**. **Division** overlaps with the refinement and **Composition** overlaps with the coarsening relationships among granules mentioned in [13, 15].

There are more concrete methods than just division and composition. Some abstract guidelines are indistinguishability, similarity, equivalence, proximity and functionality proposed by Zadeh [19, 20] and Keet [10]. Other useful mechanisms in SE include **causality**, **deduction** (attribute assignment, instantiation, and classification), **induction** or **abstraction** (value extraction, structure extraction, and meta-structure extraction).

Method 1: causality. This is to find a reason granule from a result granule or a result one from a reason one. For example, “driveIndependently(aCar)” is a result granule, “pass(roadTest)” is its reason granule. There may be multiple reasons for one result or multiple results for one reason. They can be combined by “ \wedge (and)” or “ \vee (or)”. **Method 1** can be both a division and a composition.

Method 2: deduction. It is to form more concrete granules from an abstract one. It is kind of **division**.

- **Deduction 1: attribute assignment.** This is a process to create granules by setting concrete values to the argument part of the granule having structures. For example, *collect(balloon)* is a granule, granules can be created as, *collect(greenBalloon)*, *collect(redBalloon)* and “*collect(whiteBalloon)*”.
- **Deduction 2: instantiation.** This is a process to create granules by setting all the concrete values to a granule having structures. For example, *drive(car)* is a granule, granules can be created as, *drive(FordCar)*, *drive(HondaCar)* and *drive(BMWCar)*.

- **Deduction 3: by classification.** This is to form more concrete granules having structures from an abstract one having structures. For example, *drive(vehicle)* is an abstract problem, it can be divided into several more concrete granules, such as, *drive(car)*, *drive(truck)*, *drive(bus)*, *drive(boat)* and even *drive(plane)*.

Method 3: abstraction (induction). It is to form a more abstract granule from concrete ones. It is kind of **composition**.

- **Abstraction 1: attribute extraction.** This is a process to create a granule by extracting a common attribute from values. For example, *collect(greenBalloon)*, *collect(redBalloon)* and *collect(whiteBalloon)* can be composed into a granule *collect(colorBalloon)*.
- **Abstraction 2: structure extraction.** This is a process to create a granule by extracting a group of attributes from values. For example, a granule *print(student)* can be composed from granules *print(“John Smith”, 22, “B.S.”, 92)*, *print(“Ted McDonald”, 20, “B.A.”, 87)*, and *print(“Mary Thomas”, 19, “B.B.A.”, 88)*, where *student* is a structure (*name, age, major, average grade*).
- **Abstraction 3: meta-structure extraction.** This is a process to form a more abstract granule having structures from concrete granules having structures. For example, granule *produce(food)* can be composed by *produce(bread)*, *produce(sandwich)*, *produce(soup)*, and *produce(Pisa)*.

Abstraction is the most complicated way to solve a problem. It is discovery and creation. The granules to be created may have never previously existed. Compared with abstraction, deduction is a little easier. **Abstraction** is also the key methodology for data mining [17, 18].

Definition 8: sub-problem. Suppose $p=v(a)$ is a problem. Every granule split from $v(a)$ in the forms of $v_j(a)$, $v(a_j)$, $v(a^0, a^1, \dots, a^n)$, $(v^0, v^1, \dots, v^m)(a)$, or $(v^0, v^1, \dots, v^m)(a^0, a^1, \dots, a^n)$ is called a sub-problem of p .

Definition 9: Granular Problem Solving (GrPS). GrPS is problem solving with granulation, i.e., a problem is iteratively granulated into sub-problems until all the sub-problems are final granules.

Axiom: A problem is solvable if:

- all its sub-problems are solvable when they are in “ \wedge (and)” relations; or
- one of its sub-problems is solvable when they are in “ \vee (or)” relations.

Evidently, **deduction** and **abstraction** introduce “and” relations and **causality** introduces both “and” and “or” relations. This is the axiomatic principle for “divide and conquer”.

4. GRANULATION IN SOFTWARE ENGINEERING

As for computer software, a problem is an abstract requirement description. Problems in SE are abstract, ambiguous, and sophisticated. They are much more complicated than information tables [18] and relations in a database system. To find such a solution is called SE. Granulation is the inherent method for SE. “Granular computing strategy has been extensively used in software system analysis, especially in designing classes in object-oriented analysis [5].” “Granulate and conquer” may improve “divide and conquer”, because granulation involves not only division but also composition.

Chandrasekaran defines “a design problem” by a set of functions and a technology [3]. The solution to a design problem consists of a complete specification of a set of components and their relations that together describe an artifact that delivers the functions and satisfies the constraints. Including a technology is consistent with the property that the solution set S is expanding.

Definition 10: Software Problem. A software problem p' is a special problem that is to be solved with a computer program.

The mark p' denotes a specific software problem, \mathcal{P}' the set of all software problems. Evidently, $\mathcal{P}' \subset \mathcal{P}$ is always true. For example, “build(an interactive system)” is a software problem.

Definition 11: Software System. A software system s' is a runnable program on a computer.

The mark s' denotes a specific software system and \mathcal{S}' the set of all software systems. A software system is implemented based on final granules. Evidently, $\mathcal{S}' \subset \mathcal{S}$ is always true. For example, “a runnable interactive system” is a solution of the problem “build (an interactive system)”.

Based on **Definition 11**, SE is a process to design a software system to solve a software problem. It is one kind of problem solving and a process of granulation.

From the above definitions and properties of problem solving, it is understandable why some SE projects are successful yet others fail. Following the idea of “divide (or granulate) and conquer” and the granulation process in Section 3, the methodologies proposed and applied in SE are actually combinations of granulation methods.

4.1 Structural Software Engineering

Structural Software Engineering (SSE) was the earliest method to divide and conquer since software crisis was reported in the software industry. It uses procedures (or functions, subprograms, and subroutine) to divide a complicated problem solving into sub-problems that can be handled by a single person.

Definition 12: Procedure. It can be expressed as $v'(a)$, v' is an operation and a' is an object, i.e., $a' \in O$. It is

actually in the same form of problem. The mark f denotes a specific procedure and \mathcal{F} the set of all the procedures.

In SSE, a software problem is a procedure. SSE is in fact granulation from the initial procedure $v'(a)$ that is called the main program in many programming languages. The software system created in SSE is implemented in a special programming language based on the final granules (final procedures). Broadly speaking, SSE applies all the six granulation methods. However, it only concentrates on the operation part v of a problem and the parameter part a is secondary to the operation part, i.e., it mainly uses “functionalities” to form granules.

SSE is actually to divide the operation v of a problem at first and the parameter a of the problem is then divided to follow the division of the operation. It supports the **divisions 1-3** and **compositions 1** and **2**. As for the methods of granulation, SSE mainly applies **causality** in procedure design, **deduction 1** in procedure call, and **abstractions 1** and **2** in constructing data structures [18]. Granulation by **causality** is substantially used in Logic Programming (LP) [23].

In fact, Functional Programming (FP) [23] uses the same granulation methods as SSE. The only difference is that FP uses lists as its major computation components but SSE uses procedures.

4.2 Object-Oriented Software Engineering

Object-Oriented Software Engineering (OOSE) is considered a better computing paradigm than SSE according to fundamental principles and concepts such as classes, instances, inheritance, polymorphism, and overloading. Classes are the kernel concept of OOSE [23].

Definition 13: class. A *class* is defined as $c ::= \langle n, \mathcal{D}, \mathcal{F}_c \rangle$ where, n is the identification of the class; \mathcal{D} is attributes for storing the state of an object; and \mathcal{F}_c is a set of the function definitions or implementations ($\mathcal{F}_c \subset \mathcal{F}$).

The mark c denotes a specific class, and \mathcal{C} the set of all classes. After classes are defined, objects can be redefined based on classes. To differentiate them from objects in a broad meaning, they are called *instances*.

Definition 14: instance. An instance o' is defined based on a class, i.e., $o' ::= \langle n, c, d \rangle$ where, n is the identification of the object; c is the object's class identified by the class identification or name; and d is a set of values responding to the attributes defined in its class c .

The mark o' denotes a specific instance and \mathcal{O}' the set of all the instances.

OOSE aims to divide the parameter a of a problem at first. The operation v of the problem is then divided following the division of the parameter. It is actually an iterative process of constructing a granule using both **divisions 1-3** and **compositions 1-3**. To granulate with

classes and objects in OOSE, the properties functionality, similarity, and proximity are applied.

As for the methods of granulation, OOSE applies all the **deduction** and **abstraction** methods. Classes in OOSE are actually a way of granulation by **classification**, i.e., **deduction 3**. **Deduction 3** in OOSE is to add more internal properties (connotation) from the original granule to decrease the extension (smaller granules). Creating instances from a class is granulation by **instantiation**, i.e., **deduction 2**. Designing a class from a group of objects and designing a class from a group of classes are granulation by **abstraction**.

Abstraction is the most difficult task for software engineers. Fortunately, many object-oriented programming languages establish extensive libraries of classes based on their designers' efforts on abstraction. Classes are built by collecting similar (similarities) attributes, same services or operations (functionalities) from a group of objects.

At a first glance, a class supports **composition 3**. However, current OOSE does not have a definite method to form a class by compositions. It is totally up to the experience, knowledge and wisdom of software engineers.

5. NEW TYPES OF GRANULATION

Role-Based Collaboration (RBC) is a computational thinking methodology or a problem solving paradigm that mainly uses roles as underlying mechanisms to facilitate abstraction, classification, separation of concern, dynamics, interactions and collaboration [22, 24, 25].

Based on the principles of RBC [22, 24], Role-Based Software Engineering (RBSE) is a newly proposed methodology to improve software development [26]. It introduces new granulation mechanisms not applied in OOSE in the sense of problem solving: roles, agents, environments and groups. By RBSE, an iterative process of granulation in **division**, **compositions**, **causality**, **deduction**, and **abstraction** are established.

Definition 15: Role. A *role* is defined as $r ::= \langle n, I, \mathcal{A}_o, \mathcal{A}_p, \mathcal{R}_x, O_a \rangle$, where, n is the identification of the role; $I ::= \langle \mathcal{M}_{in}, \mathcal{M}_{out} \rangle$ denotes a set of messages, where \mathcal{M}_{in} expresses the incoming messages to the relevant agents, and \mathcal{M}_{out} expresses a set of outgoing messages or message templates to roles, i.e., $\mathcal{M}_{in}, \mathcal{M}_{out} \subset \mathcal{M}$; \mathcal{A}_c a set of agents who are currently playing this role; \mathcal{A}_p a set of agents who are qualified to play this role; \mathcal{A}_o a set of agents who used to play this role; \mathcal{R}_x a set of roles interrelated with it; and O_a a set of objects to be accessed by the agents playing this role.

The mark r denotes a specific role and \mathcal{R} the set of all the roles. Roles are considered as an abstraction and decomposition mechanism related to agents. When an agent plays a role, it accepts messages, provides services and sends out requests related to its role. A role constitutes

a part of an agent's behavior that is obtained by considering the interactions of that role and hiding all other interactions.

Definition 16: Agent. An *agent* is defined as $a_g ::= \langle n, c_a, s, d, r_c, \mathcal{R}_p, \mathcal{R}_o, G_a \rangle$, where, n is the identification of the agent; c_a a special class that describes the common properties of users; s the qualifications of the agent; d the credit of the agent; r_c a role that the agent is currently playing; \mathcal{R}_p a set of roles that the agent is qualified to play ($r_c \notin a.\mathcal{R}_p$); \mathcal{R}_o a set of roles that the agent played before; and G_a a set of groups that the agent belongs to [24].

The mark a_g denotes a specific agent and \mathcal{A}_g the set of all the agents.

In RBSE, roles are split into or composed from (v^0, v^1, \dots, v^m) and agents are split into or composed from (a^0, a^1, \dots, a^n) to support **composition 3** of granulation, i.e., an agent playing a role or a role being played by an agent forms a granule.

Roles are used to show common interfaces among agents in interactions. Agents are special objects being able to play roles. They are different from objects in that they play roles whereas objects do not. Roles are good mechanisms to assist granulation. Compared with other mechanisms, roles and agents can be applied to more complicated real-world problems such as system analysis and system design, because these "divide and conquer" tasks are much more complicated than those in databases. A new method of granulation can be introduced, where *rolification* means the act or the result of designing roles.

Method 4: rolification. Granulation by rolification is to use *roles* or the common interfaces of requests and services (similarities and functionalities) and *agents* or the relevant special objects to create new granules. In this method, a granule can be expressed as $r(a_g)$, where r comes from (v^0, v^1, \dots, v^m) and a_g from (a^0, a^1, \dots, a^n) .

Rolification is a granulation method on top of classes. Roles can be further refined into more operable granulations: **current roles**: a granule can be formed by playing the same current role r_c by a group of agents; **potential roles**: a granule can be formed by the same potential roles \mathcal{R}_p attached by a group of agents; and **past roles**: a granule can be formed by the same past roles \mathcal{R}_o attached by a group of agents. Based on these roles, more relationships among granules can be established. A case study of such method can be found in [21].

6. RELATED WORK

Chandrasekaran [3] proposes a task (i.e. software problem) structure for design by analyzing a general class of methods. He shows that there is no one ideal method for designs and good design problem solving is a result of recursively selecting methods based on a number of criteria, including knowledge availability. Yao and Zhong

[18] propose several methods to form granules: by equality of attribute values; by equivalence of attribute values; and by similarity of attribute values. Their methods can be applied in data analysis and data mining. Liu [8] presents his formal methods in Artificial Intelligence (AI) reasoning with granules. He defines the elementary granule of decision rules as a pair $(\varphi, d(\varphi))$ in which φ is the syntax and $d(\varphi)$ is the semantics. Han and Dong [5] discuss the architecture of granular computing models, strategies, and applications. They investigate granular computing from the perspectives of SE, including requirement specification and analysis, system analysis and design, algorithm design, structured programming, software testing, and system deployment. Yao [15] examines the basic principles and issues of granular computing, analyzes the tasks of granulation and computing with granules, discusses the construction, interpretation, and representation of granules, as well as principles and operations of computing and reasoning with granules. Yao [13] discusses granular computing theory from the perspective of information granulation and granular relationships. These relationships are good supplements in granular problem solving.

7. CONCLUSIONS

The development of SE methodologies may be based on different forms and methods of granulation. Granulation is one of the foundations of problem solving and SE. Rolification is a new type of granulation in GrPS.

To improve SE methodologies should be to introduce indistinguishability and proximity, because current SE methodologies have not yet introduced these properties. Roles are also a potential granulation mechanism to be applied in data mining. More case studies and evidences are required to collect to show more merits of roles.

ACKNOWLEDGMENTS

This research is in part supported by National Sciences and Engineering Research Council of Canada (NSERC: 262075-06). Thanks also go to Mike Brewes of Nipissing University for proofreading this article.

REFERENCES

- [1] Bargiela, A. and Pedrycz, W., "The roots of granular computing", *Proc. of the IEEE Int'l Conf. on Granular Computing*, May 2006, pp. 806- 809.
- [2] Capra, F. *The Web of Life*, Anchor Books, New York, 1997.
- [3] Chandrasekaran, B., "Design problem solving: a task analysis", *AI Magazine*, vol. 11, no., 4, 1990, pp. 59-71.
- [4] Dictionary.com, <http://dictionary.reference.com>, 2008.
- [5] Han, J.; Dong, J., "Perspectives of granular computing in Software Engineering," *Proc. of the IEEE Int'l Conf. on Granular Computing*, Silicon Valley, CA, USA, Nov. 2007(GrC'07), pp. 66-71.
- [6] Lin, T.Y., "Granular computing: examples, intuitions and modeling", *Proc. of IEEE Int'l Conf. on Granular Computing*, July 2005, Beijing, China (GrC'05), pp. 40-44.
- [7] Lin, T.Y., "Granular computing: a problem solving paradigm", *Proc. of the IEEE Int'l Conf. on Fuzzy Systems*, May 22-25, 2005, Reno, Nevada, USA, pp. 132-137.
- [8] Liu, Q., "Granules and reasoning based on granular computing", *Proc. of the 16th Int'l Conf. on Developments in Applied Artificial Intelligence, Laughborough, UK, 2003, Lecture Notes In Computer Science*, vol. 2718, pp. 516 – 526.
- [9] Louie, E., Lin, T.Y. and Hu, X., "Analysis of using granules to find association rules", *GrC'05*, pp. 556-560.
- [10] Keet, C. M., "Granulation with indistinguishability, equivalence, or similarity", *GrC'07*, pp. 11-16.
- [11] Pedrycz, W., "Granular computing: an introduction", *Proc. of the IFSA World Congress and 20th NAFIPS Int'l Conf.*, July 2001, Vancouver, BC, Canada, vol. 3, pp. 1349-1354.
- [12] Yao, J.T. and Yao, Y. Y., "Information granulation for web based information retrieval support systems", *Proc. of SPIE, Vol. 5098*, April 2003, Orlando, FL, USA, pp. 138-146.
- [13] Yao, J.T., "Information granulation and granular Relationships", *GrC'05*, pp. 326- 329.
- [14] Yao, J. T., "A Ten-Year Review of Granular Computing", *GrC'07*, pp734-739.
- [15] Yao, Y.Y., "A partition model of granular computing", *LNCS Transactions on Rough Sets I*, vol. 3100, 2004, pp. 232-253.
- [16] Yao, Y.Y. "The art of granular computing", *Proc. of the Int'l Conf. on Rough Sets and Emerging Intelligent Systems Paradigms*, LNAI 4585, 2007, pp. 101-112.
- [17] Yao, Y.Y., "Perspectives of granular computing", *GrC'05*, pp. 85-90.
- [18] Yao, Y.Y. and Zhong, N., "Granular computing using information tables", *Lin, T.Y., Yao, Y.Y. and Zadeh, L.A. (Eds.), Data Mining, Rough Sets and Granular Computing*, Physica-Verlag, Heidelberg, 2002, pp. 102-124.
- [19] Zadeh, L.A. "Some reflections on soft computing, granular computing, and their roles in the conception, design and utilization of information/intelligent systems", *Soft Computing*, vol. 2, 1998, pp. 23-25.
- [20] Zadeh, L.A. "Towards a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic", *Fuzzy Sets and Systems*, vol. 19, 1997, pp. 111-127.
- [21] Zhu, H., "Improving Object-Oriented Analysis with Roles", *Proc. of the IEEE Int'l Conf. on Cognitive Informatics*, Lake Tahoe, CA, USA, Aug. 2007, pp. 430-439.
- [22] Zhu, H., "The role mechanisms in collaborative systems", *Int'l J. of Production Research*, vol. 44, no. 1, 2006, pp. 181-193.
- [23] Zhu, H. and Zhou, M.C., *Object-Oriented Programming with C++*, Tsinghua Univ. Press, 2006.
- [24] Zhu, H. and Zhou, M.C., "Role-based collaboration and its kernel mechanisms", *IEEE Trans. on Systems, Man and Cybernetics, Part C*, vol. 36, no. 4, 2006, pp. 578-589.
- [25] Zhu, H. and Zhou, M.C., "Supporting software development with roles", *IEEE Trans. on Systems, Man and Cybernetics, Part A*, vol. 36, no. 6, 2006, pp. 1110-112.