

---

## **Granular problem solving and its applications in software engineering**

---

**Haibin Zhu**

Department of Computer Science and Mathematics,  
Nipissing University,  
100 College Drive, North Bay,  
Ontario, P1B 8L7, Canada  
E-mail: 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 of this paper is establishing the foundations of granular problem solving and clarifying the relationships between granular problem solving and software engineering.

**Keywords:** problem solving; granular computing; GrC; granule; granulation; role; agent.

**Reference** to this paper should be made as follows: Zhu, H. (xxxx) 'Granular problem solving and its applications in software engineering', *Int. J. Granular Computing Rough Sets and Intelligent Systems*, Vol. X, No. Y, pp.000–000.

**Biographical notes:** Haibin Zhu is an Associate Professor of the Department of Computer Science and Mathematics, Nipissing University, Canada and has published five books, one book chapter and more than 80 research papers. He is a Senior Member of IEEE and a member of ACM. He is serving and served as Co-chair of the Technical Committee of Distributed Intelligent Systems of IEEE SMC Society, Guest (Co-)Editor for three special issues of prestigious journals, PC Vice Chair, Poster Co-chair and Publicity Co-chair for many IEEE conferences. He is the recipient of the Best Paper Award from ISPE/CE'04 and two IBM EIG awards.

---

### **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 (Bargiela and

Pedrycz, 2006) and believed to be a new type of problem solving paradigm (Lin, 2005a, 2005b; Pedrycz, 2001; Yao and Yao, 2003; Yao, 2005; Yao, 2007; Yao, 2004; Yao, 2007; Yao, 2005). 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 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 analyse 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 and clarifies the properties of problem solving. Section 3 defines granules and granulation and analyses 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 defining two new concepts 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

Concepts are the first step of scientific research. They provide a basis in the formation of new models. 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. It is necessary to define fundamental and abstract concepts and to understand the basic properties of problem solving.

Dictionaries (Dictionary.com, 2008) 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 is, a real-world set of objects is involved. “A real-world problem normally consists of a web of interacting and interrelated parts” (Capra, 1997).

*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 languages,  $v$  is a verb and  $a$  is a noun. The mark  $p$  denotes a specific problem and  $P$  the set of all the problems. Evidently,  $P \subset O$  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  $S$  the set of all the solutions. Evidently,  $S \subset O$  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  $p \in P$  can be defined as an activity to find a map  $\langle p, s \rangle$  where  $s \in S$ . The mark  $s$  denotes a specific solution and  $S$  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  $S$  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)’; ‘the homework done’ is a concrete solution but ‘learn, understand and do’ is an abstract one; the problem ‘callOnAPlane(aFriend)’ had no solution in 1960s but has one now; mathematical problems have solutions in different forms, such as numbers, formula, equations, tables and graphs; health problems have solutions in the forms of pills, surgeries and transplants and software problems have solutions in the form of software systems.

### 3 Granulation

GrC is a form of problem solving. Granulation involves the fundamental concepts and mechanisms related to general problem solving. Granulation or computing with granules, can be considered as knowledge processing. Granules can be taken as the basic entities of knowledge (Lin, 2005a). “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” (Yao and Zhong, 2002). Granulation is a process of decomposition from whole to parts (Louie et al., 2005; Pedrycz, 2001; Yao and Yao, 2003; Yao, 2005; Yao, 2007; Yao, 2004; Yao, 2007; Yao, 2005).

Granulation in fact involves both composition and division. A granule is a group of objects that are drawn together based on some criteria (Lin, 2005a, 2005b). The definitions in Section 2 are actually a process of granulation: building one set from another. 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 solution easier.

*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,  $P_g$  the set of all granules. Evidently,  $P_g \subset P$  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:*  $v(a) \xrightarrow{d} \{v_i(a) \mid m > 1, 0 \leq i < m\}$ . 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:*  $v(a) \xrightarrow{d} \{v(a_j) \mid n > 1, 0 \leq j < n\}$ . 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:*  $v(a) \xrightarrow{d} \{v_i(a_j) \mid m > 1, n > 1, 0 \leq i < m, 0 \leq j < n\}$ . 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  $a_j$ .

Let us consider a problem with granulation. 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  $x(+ (12, 35), - (34, 23))$ ) is a problem. It can be divided into several simple problems, such as  $12 + 35$ ,  $34 - 23$  and  $47 \times 11$ , 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. The process of problem solution does not always require splitting. Sometimes, composition is required.

From the viewpoint of composition, three forms of compositions are as follows corresponding to the three forms of divisions respectively, where, ' $\xleftarrow{c}$ ' is used to express 'is composed from':

- *Composition 1*:  $\{(v^0, v^1, \dots, v^{m'}) (a) \mid m > 1, 0 \leq m' < m\} \xleftarrow{c} \{v_i(a) \mid m > 1, 0 \leq i < m\}$ .  
It expresses that a new operation  $(v^0, v^1, \dots, v^{m'})$  is composed from  $m'$  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*:  $\{v(a^0, a^1, \dots, a^{n'}) \mid n > 1, 0 \leq n' < n\} \xleftarrow{c} \{v(a_j) \mid n > 1, 0 \leq j < n\}$ .  
It expresses that a new parameter  $(a^0, a^1, \dots, a^{n'})$  is composed from  $n'$  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*:  $\{(v^0, v^1, \dots, v^{m'}) (a^0, a^1, \dots, a^{n'}) \mid m > 1, 0 \leq m' < m, n > 1, 0 \leq n' < n\} \xleftarrow{c} \{v_i(a_j) \mid m > 1, n > 1, 0 \leq i < m, 0 \leq j < n\}$ . It expresses that a new operation  $(v^0, v^1, \dots, v^{m'})$  is composed from  $m'$  partial operations of the  $m$  partial operations  $v_i$  and a new parameter  $(a^0, a^1, \dots, a^{n'})$  is composed from the  $n'$  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’, 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(tyre)’ and ‘build(auxiliary)’. This process is *Division 2*. Granules ‘buy(software)’, ‘reuse(software)’, ‘makeBySelf(software)’ and ‘borrow(software)’ can be composed into a new granule ‘build(software)’. It is a process of *Composition 1*.

*Division* overlaps with the refinement and *composition* the coarsening relationships among granules mentioned in Yao (2005) and Yao (2004).

There are more methods than just division and composition. Some abstract guidelines are indistinguishability, similarity, equivalence, proximity and functionality proposed by Zadeh (1997, 1998) and Keet (2007). Other useful mechanisms in SE include *causality*, *deduction* (attribute assignment, instantiation and classification), *induction* or *abstraction* (value extraction, structure extraction and meta-structure extraction).

To be more operable, concrete methods can be categorised as follows:

*Method 1 Causality*. This is a process 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)’. Therefore, *Method 1* can be both a division and a composition.

*Method 2 Deduction.* It is a process to form more concrete granules from an abstract granule. It is a 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: Classification.* This is a process to form more concrete granules having structures from an abstract granule 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 a process to form a more abstract granule from concrete granules. 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(colourBalloon)*.
- *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, because an abstract thing has been created by concrete ones. *Abstraction* is also the key methodology for data mining (Yao, 2005; Yao and Zhong, 2002).

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

*Definition 9: 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 SE

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 (Yao and Zhong, 2002) and relations in a database system. A solution is a computer program. To find such a solution is called SE. SE is one kind of problem solving.

Granulation is in fact the inherent method for SE. “GrC strategy has been extensively used in software system analysis, especially in designing classes in object-oriented analysis” (Han and Dong, 2007). ‘Granulate and conquer’ may improve ‘divide and conquer’, because granulation involves not only division but also composition.

Chandrasekaran (1990) defined ‘a design problem’ by a set of functions and a technology. He notices that the solution to a design problem consists of a complete specification of a set of components and their relations that together describe an artefact that delivers the functions and satisfies the constraints. Evidently, that his definition includes 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.

The mark  $p'$  denotes a specific software problem,  $P'$  the set of all software problems. Evidently,  $P' \subset P$  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  $S'$  the set of all software systems. A software system is implemented based on final granules. Evidently,  $S' \subset S$  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 (SSE)

SSE was the earliest method to divide and conquer since software crisis was reported in the software industry. It uses procedures (also called functions, sub-programs and sub-routine) 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  $F$  the set of all the procedures.

In SSE, a software problem is a procedure. Procedures are used to granulate smaller and simpler processes from a larger and more complicated process. 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 *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 (Yao and Zhong, 2002). Granulation by *causality* is substantially used in logic programming (LP) (Zhu and Zhou, 2006a).

In fact, functional programming (FP) (Zhu and Zhou, 2006a) 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 (OOSE)

OOSE is considered a better computing paradigm than SSE according to fundamental principles (Zhu and Zhou, 2006a): “everything in the world is an object; every system is composed of objects and certainly a system is also an object; and the evolution and development of a system is caused by the interactions among the objects inside or outside the system”. Continuing to the details of OOSE, classes, instances, inheritance, polymorphism, overloading and overriding are introduced (Zhu and Zhou, 2006a). Classes are the kernel concept of OOSE.

*Definition 13: Class.* A class is defined as  $c ::= \langle n, D, F_c \rangle$  where

- $n$  is the identification of the class
- $D$  is attributes for storing the state of an object
- $F_c$  is a set of the function definitions or implementations ( $F_c \subset F$ ).

The mark  $c$  denotes a specific class and  $C$  the set of all classes. After classes are defined, objects can be redefined based on classes. To be differentiated 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
- $d$  is a set of values responding to the attributes defined in its class  $c$ .

The mark  $o'$  denotes a specific instance and  $O'$  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, *Deduction 3*. The granulation by *classification* 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, and same services or operations (functionalities) from a group of objects.

At 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 the software engineers. The next section introduces new mechanisms to improve this situation.

## 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. It encompasses computational concepts, methods, models, algorithms and tools. Based on roles, RBC is an emerging methodology used to facilitate an organisational structure, provide orderly system behaviour and consolidate system security for both human and non-human entities that collaborate and coordinate their activities with or within systems. RBC is a way to understand the complexity in natural, built and social systems, because RBC enables the design, synthesis and control of novel complex systems (Zhu, 2006; Zhu and Zhou, 2006b, 2006c).

Based on the principles of RBC (Zhu, 2006; Zhu and Zhou, 2006b), role-based software engineering (RBSE) is a newly proposed methodology to improve software development by taking software development as a collaborative activity (Zhu and Zhou, 2006c). It introduces new granulation mechanisms not applied in OOSE in 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  $r ::= \langle n, I, s_r, d_r, A_c, A_p, A_o, R_x, O_a \rangle$  where,

- $n$  is the identification of the role
- $I ::= \langle M_{in}, M_{out} \rangle$  denotes a set of messages, where  $M_{in}$  expresses the incoming messages to the relevant agents and  $M_{out}$  expresses a set of outgoing messages or message templates to roles, i.e.,  $M_{in}, M_{out} \subset M$
- $s_r$  is the qualification requirement of this role
- $d_r$  is the credit requirement of this role
- $A_c$  is a set of agents who are currently playing this role
- $A_p$  is a set of agents who are qualified to play this role
- $A_o$  is a set of agents who used to play this role
- $R_x$  is a set of roles interrelated with it
- $O_a$  is a set of objects that can be accessed by the agents playing this role.

The mark  $r$  denotes a specific role and  $R$  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 behaviour that is obtained by considering the interactions of that role and hiding all other interactions.

*Definition 16: Agent.* An agent is defined as  $a_g ::= \langle n, c_a, s, d, r_c, R_p, R_o, G_a \rangle$ , where

- $n$  is the identification of the agent
- $c_a$  is a special class that describes the common properties of users
- $s$  is the qualifications of the agent
- $d$  is the credit of the agent
- $r_c$  means a role that the agent is currently playing. If it is empty, then this agent is free
- $R_p$  means a set of roles that the agent is qualified to play ( $r_c \notin a, R_p$ )
- $R_o$  means a set of roles that the agent played before
- $G_a$  means a set of groups that the agent belongs to (Zhu and Zhou, 2006b).

The mark  $a_g$  denotes a specific agent and  $A_g$  the set of all the agents.

In RBSE, roles are split or composed from  $(v^0, v^1, \dots, v^m)$  and agents are split 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. They can

play the same role as other mechanisms in granulations, such as, classes, clusters, objects, etc. 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  $(a^0, a^1, \dots, a^{n'})$ .

It is a granulation method on the top of classes. This method can be further refined into more operable granulations: by *current roles*: a granule can be formed by playing the same current role  $r_c$  by a group of agents; by *potential roles*: a granule can be formed by the same potential roles  $R_p$  attached by a group of agents and by *past roles*: a granule can be formed by the same past roles  $R_o$  attached by a group of agents. Based on these roles, more relationships among granules can be established.

### 5.1 A case analysis with roles and agents

System analysis can be taken as a kind of data mining. Roles have shown their merits with higher efficiency than classes (Zhu, 2007). In RBSE, agents are created based on nouns that can be taken as roles and classes are created based on nouns that cannot be roles. These classes do not have to consider the combinations of nouns as in traditional OOSE. The  $n$  nouns can be separated into  $m$  roles and  $n-m$  non-roles. Here,  $m$  and  $n-m$  are significantly less than  $2^n - 1$  (a combination of  $n$  nouns). There may be a few more than  $m$  based on the domain knowledge (DK) because many requirement descriptions do not include all the roles involved. For example, in a description with 111 words of purchase management, there are 12 nouns. The possible number of classes is  $2^{12} - 1 = 4,095$ . However, RBSE will require only five roles (two from the description and three from DK) and ten classes since two of the 12 nouns have been used for roles (Zhu, 2007). Five roles specify five sets of interfaces for designing agents. This can greatly save the effort of system analysis.

## 6 Related work

Chandrasekaran (1990) proposes a task (i.e., software problem) structure for design by analysing a general class of methods. The task is structured by identifying a range of methods. For each method, the required knowledge and the subtasks that it sets up are identified. This recursive analysis provides a framework to understand design problem solving as specific combinations of tasks, methods and subtasks. The analysis 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 (2002) propose several methods to form granules: by equality of attribute values; by equivalence of attribute values and by similarity of attribute values. They introduce information tables and a decision logic language to facilitate the above three methods. Based on information tables, granules can be constructed by combining objects categorised in the above three criteria. In their methods, a granule is a concept such that each element in the granule is an instance of the concept. Their methods can be applied in data analysis and data mining, i.e., extracting knowledge by forming granules that are concepts.

Liu (2003) 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  $\varphi$  is the syntax and  $d(\varphi)$  is the semantics. Based on his definition of granules, he proposes a new reasoning model that can be applied in expert systems.

Han and Dong (2007) discuss the architecture of GrC models, strategies and applications. They investigate GrC from the perspectives of SE, including requirement specification and analysis, system analysis and design, algorithm design, structured programming, software testing and system deployment.

Yao (2004) examines the basic principles and issues of GrC, analyses 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. Based on his analyses, he proposes a partition model of GrC by combining, reformulating and reinterpreting notions and results from several related fields, such as, theories of granularity, abstraction and generalisation, partition models of databases, coarsening and refining operations, set approximations and the quotient space theory for problem solving.

Yao (2005) discusses GrC theory from the perspective of information granulation and granular relationships. He specifies several granule relationships such as intra-relationship, refinement and coarsening, partitions and coverings, partial ordering, IS-A relationships, similarity relationship and fuzzy relationships. These relationships are good supplements in GrPS.

## 7 Conclusions

From the above discussion, it can be seen that the development of SE methodologies maybe based on different forms and methods of granulation. Granulation is one of the foundations of problem solving and SE. Rolification is proposed as a new type of granulation in GrPS.

The future direction of improving SE methodologies should be to introduce indistinguishability and proximity, because current SE methodologies have not yet introduced these properties. From the definitions of RBC and its model E-CARGO (Zhu, 2006; Zhu and Zhou, 2006b), two other components (environments and groups) are potential granulation methods. It is an interesting topic to be investigated. Roles are also potential granulation mechanism to be applied in data mining. More case studies and evidences are required to collect to show more merits of roles.

## Acknowledgements

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

## References

- Bargiela, A. and Pedrycz, W. (2006) 'The roots of granular computing', *Proc. of the IEEE Int'l Conf. on Granular Computing*, pp.806–809.
- Capra, F. (1997) *The Web of Life*, Anchor Books, New York.
- Chandrasekaran, B. (1990) 'Design problem solving: a task analysis', *AI Magazine*, Winter, Vol. 11, No. 4, pp.59–71.
- Dictionary.com (2008) available at <http://dictionary.reference.com>.
- Han, J. and Dong, J. (2007) 'Perspectives of granular computing in software engineering', *IEEE Int'l. Conf. on Granular Computing, 2007(GrC 2007)*, 2–4 November, pp.66–71.
- Keet, C.M. (2007) 'Granulation with indistinguishability, equivalence or similarity', *Proc. of the IEEE International Conference on Granular Computing*, Silicon Valley, CA, USA, pp.11–16.
- Lin, T.Y. (2005a) 'Granular computing: a problem solving paradigm', *Proc. of the IEEE Int'l Conf. On Fuzzy Systems*, 22–25 May 2005, Reno, Nevada, USA, pp.132–137.
- Lin, T.Y. (2005b) 'Granular computing: examples, intuitions and modeling', *Proc. of IEEE Int'l Conf. on Granular Computing*, Beijing, China, pp.40–44.
- Liu, Q. (2003) 'Granules and reasoning based on granular computing', *Proc. of the 16th Int'l Conf. on Developments in Applied Artificial Intelligence Lecture Notes in Computer Science*, Laughborough, UK, Vol. 2718, pp.516–526.
- Louie, E., Lin, T.Y. and Hu, X. (2005) 'Analysis of using granules to find association rules', *Proc. of the IEEE Int'l Conf. on Granular Computing*, Beijing, China, Vol. 2, pp.556–560.
- Pedrycz, W. (2001) 'Granular computing: an introduction', *Proc. of the IFSA World Congress and 20th NAFIPS International Conference*, Vancouver, BC, Canada, Vol. 3, pp.1349–1354.
- Yao, J.T. (2005) 'Information granulation and granular relationships', *Proc. of the IEEE Int'l Conf. on Granular Computing*, Beijing, China, Vol. 1, pp.326–329.
- Yao, J.T. (2007) 'A ten-year review of granular computing', *Proc. the IEEE Int'l Conf. on Granular Computing*, Silicon Valley, CA, USA, pp.734–739.
- Yao, J.T. and Yao, Y.Y. (2003) 'Information granulation for web based information retrieval support systems', *Proc. of SPIE*, Orlando, FL, USA, Vol. 5098, pp.138–146.
- Yao, Y.Y. (2004) 'A partition model of granular computing', *LNCS Transactions on Rough Sets I*, Vol. 3100, pp.232–253.
- Yao, Y.Y. (2005) 'Perspectives of granular computing', *Proceedings of 2005 IEEE Int'l Conf. on Granular Computing*, Beijing, China, Vol. 1, pp.85–90.
- Yao, Y.Y. (2007) 'The art of granular computing', *Proc. of the Int'l Conf. on Rough Sets and Emerging Intelligent Systems Paradigms, LNAI 4585*, pp.101–112.
- Yao, Y.Y. and Zhong, N. (2002) 'Granular computing using information tables', in Lin, T.Y., Yao, Y.Y. and Zadeh, L.A. (Eds.): *Data Mining, Rough Sets and Granular Computing*, pp.102–124, Physica-Verlag, Heidelberg.
- Zadeh, L.A. (1997) 'Towards a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic', *Fuzzy Sets and Systems*, Vol. 19, pp.111–127.
- Zadeh, L.A. (1998) 'Some reflections on soft computing, granular computing and their roles in the conception, design and utilization of information/intelligent systems', *Soft Computing*, Vol. 2, pp.23–25.

- Zhu, H. (2006) 'The role mechanisms in collaborative systems', *Int'l J. of Production Research*, Vol. 44, No. 1, pp.181–193.
- Zhu, H. (2007) 'Improving object-oriented analysis with roles', *Proc. of the IEEE Int'l Conf. on Cognitive Informatics*, Lake Tahoe, CA, USA, pp.430–439.
- Zhu, H. and Zhou, M.C. (2006a) *Object-Oriented Programming with C++: A Project-Based Approach*, Tsinghua University Press.
- Zhu, H. and Zhou, M.C. (2006b) 'Role-based collaboration and its kernel mechanisms', *IEEE Trans. on Systems, Man and Cybernetics, Part C*, Vol. 36, No. 4, pp.578–589.
- Zhu, H. and Zhou, M.C. (2006c) 'Supporting software development with roles', *IEEE Trans. on Systems, Man and Cybernetics, Part A*, Vol. 36, No. 6, pp.1110–1123.