

Perspectives of Granular Computing in Software Engineering

Jianchao Han

*Department of Computer Science
California State University Dominguez Hills
jhan@csudh.edu*

Jing Dong

*Department of Computer Science
University of Texas at Dallas
jdong@utdallas.edu*

Abstract

Granular computing is not only a computing model for computer-centered problem solving, but also a thinking model for human-centered problem solving. Some authors have presented the structures of such kind models and investigated various perspectives of granular computing from different application points of views. In this paper we discuss the architecture of granular computing models, strategies, and applications. Especially, the perspectives of granular computing in various aspects and phases of software engineering are presented, including requirement specification and analysis, system analysis and design, algorithm design, structured programming, software testing, and system deployment

1. Introduction

Granular computing (GrC), as a general computing paradigm of problem solving, has been received much attentions recently, although its basic ideas and principles have been studied extensively in various research communities and application domains for a long time in explicit or implicit fashions. Zadeh [17] first introduced the notion of information granulation in 1979 and suggested that fuzzy set theory may find potential applications in this respect, which pioneers the explicit study of granular computing. With the concept of his information granulation, Zadeh further presented granular mathematics [18]. Pawlak proposed the rough set theory to deal with inexact information by using rough sets to approximate a crisp set in 1982 [9], and investigated the granularity of knowledge from the point of view of rough set theory [10]. Hobbes [5] presented a theory of granularity as the base of knowledge representation, abstraction, heuristic search, and reasoning in 1985. In his theory the problem world is represented as various grains and only interesting ones are abstracted to learn concepts.

The conceptualization of the world can be performed at different granularities and switched between granularities. In 1992, Giunchiglia and Walsh presented a theory of abstraction to improve the conceptualization of granularities [3]. Lin suggested the term “granular computing” to label this growing research field in 1997 [7]. Yao investigated the trinity model of granular computing from three perspectives: philosophy, methodology, and computation [15].

In the past decade, different granular computing models have been conducted in various aspects and applied in various application domains, including machine learning, data mining, bioinformatics, e-Business, network security, high-performance computing and wireless mobile computing, etc. The essence of these models has been addressed by researchers to build efficient computational algorithms for handling huge amounts of data, information and knowledge. The objectives of these computation models are computer-centered and mainly concern the efficiency, effectiveness, and robustness of using granules such as classes, clusters, subsets, groups and intervals in problem solving. In recent years, some researchers have investigated the granular computing paradigm from perspectives of philosophy, cognitive science, and human thinking [14, 15] as well as the general strategies of interactions between granules [13] and operations on granule coverings [12].

We realize that the basic concept of granular computing has been extensively applied in many problem solving disciplines with different formats either consciously or unconsciously by human beings. For instance, Yao [16] scrutinize the structured writing with granular computing strategies and demonstrates that by consciously using granular computing strategies and heuristics of granular computing, a scientist has a better chance to produce a clear, structured, and comprehensible document.

In this paper, we review granular computing strategies from the perspective of different phases of software engineering, especially. By investigating these strategies of granular computing in various phrases of software engineering, we expect that

software developers can consciously apply them in the process of software development and gain benefit of high-quality software products. In Section 2, the basic concepts and components of granular computing are introduced. In Section 3, the software development process is investigated from the perspective of granular computing, including requirement analysis, use cases, system design, program structure, algorithm design, testing, and deployment. Section 4 is the conclusion.

2. Basic Concepts and Components of Granular Computing

A formal, precise and uncontroversial definition and a framework of granular computing do not exist. However, granular computing has been generally understood as a paradigm of problem solving with granules as the basic elements and granulation as the primary operation. Besides granules and granulation, relationships among granules and computations with granules constitute the main ingredients of granular computing. In this section, we summarize these concepts and components.

Granules: Basically, granules are used to represent the elementary units of a complex system that is considered, and collectively provide a representation of the unit with respect to a particular level of granularity. Each granule may reflect a specific aspect of the problem or form a portion of the system domain. In the real-world problem solving, we may need to consider the levels of detail of a system. On the different levels, granules may represent different units. Using object-oriented software as an example, one can see that the software system is considered as a whole, its ingredients are a collection of classes. However, when a class is developed, its granules are instance fields, methods, and constructors/destructors. All granules form a hierarchy structure. The hierarchy structure consists of multiple levels and granules are located at different levels. Varieties of hierarchy structures reflect different granulation criteria and granular views.

Granulation: Granulation is to granulate a complex problem into granules, including the construction, representation, and interpretation of granules. It concerns the procedures for constructing granules. Granulation is a very natural concept and appears almost everywhere in different names. Granulation includes two important aspects [14, 15]. **Granulation methods** deal with how a problem or a set of elements is granulated into granules. Top-down method starts from the problem space as a whole and partitions the problem into sub-spaces, which, in turn, are partitioned again, to construct desired granules; while bottom-up methods attempts to put individual

elements together to form blocks, which, in turn, are united to build granules at expected levels. **Granulation criteria** determines whether a granule should be granulated into smaller granules for top-down construction, or whether different elements/granules should be put together to form a larger granule for bottom-up construction. Partition vs. covering should be distinguished. A partition of the universe is a collection of pair-wise disjoint nonempty granules, and the union of these granules forms the whole universe. However, a granular covering of the universe is a collection of granules that cover the whole universe, where those granules may overlap. Operations on partitions and coverings have been investigated in literature [12, 13, 15].

Granular Relationships: Relationships between granules can be interpreted mathematically as set relations or philosophically as concept relations. From another point of view, each granule may be interpreted as an implication rule or a set of attribute-value pairs, e.g., in classification and clustering analysis. Relationships between granules can also be represented as binary relations and interpreted as dependency, inheritance, etc. [13] Generally speaking, there are two types of relationships among granules: interrelationship and intrarelationship. The former is the basis of grouping small objects together to construct a larger granule based on similarity, indistinguishability, and functionality, while the latter concerns the decomposition of a large granule into smaller units and the interactions between components of a granule as well. A granule is a refinement of another granule if it is contained in the latter. Similarly, the latter is called a coarsening of the former. This relationship functions like set-containment in the set-based domains.

Computation with Granules: Computation with granules is to compute and reason the granular entities in terms of their relationships. Different operations can be performed on granules for various computation and reasoning tasks and purposes. These operations can be categorized as either computations within granules or computations between granules. Computations within granules are usually performed on the intrarelationships of granules. Typical computations in this category include finding characterization of granules, e.g. membership functions of fuzzy granules [17, 18]; inducing rules from granules, e.g. classification rules that characterize the classes of objects; forming concepts that granules entail. On the other hand, computations between granules usually operate on the interrelations between granules, transformations from one granule to another, clustering granules, and dividing granules.

3. Perspectives in Software Engineering

Granular computing as a machine-centered computation model has been extensively studied in machine learning, data mining, fuzzy set theory, rough set theory, but not paid enough attention as a human-centered thinking paradigm, although some research on this line has been conducted. From the previous section, one may see that basic components and concepts of granular computing, such as granules, partitions, hierarchies, are actually applied in problem solving process in our daily life. Granular computing provides the infrastructure for data, information and knowledge engineering as well as uncertainty management. In software engineering, the strategies of granular computing are also broadly used in all phases. Granulate-and-conquer is a softer version of classical divide-and-conquer strategy. A very common technique used in the classical “non-partitioning” recursive call is dynamic programming. Functional decomposition is to partition user requirement into granules (functions). Structured programming is to organize the computer programs as a collection of modules (procedures, functions, routines). The intrarelations among these granules are based on their ingredients such as input parameters, output results, executable statements, whereas the interrelationships involve the module interfaces and procedure calls. In object-oriented programming, granules are classes, intrarelations are the interactions between components of classes such as instance fields, methods and constructors in the Java language; and the interrelationships are class inheritance, aggregation, association, delegation, dependency, etc.

In this section, we scrutinize the different phases of modern software engineering process based on object-oriented technologies, including requirement analysis, system analysis, design, and implementation.

3.1. User Requirements

In the object-oriented methodology, user requirements analysis involves two main aspects: identifying business actors and use cases. Both aspects can be conducted with granular computing paradigm. However, only identifying use cases will be investigated in this subsection.

Basically, each use case is a snippet of the business and may involve two-way communication between actors. Consider the business requirement as the problem domain, and the use cases will be the granules. Although there’s no set rule for deciding how to granulate the business into use cases and most

analysts partition the business process into use cases based on common sense, business logic, and their experience. The basic thinking lines of granular computing can be followed.

Granules: Use cases.

Granulation method: top-down decomposition for complete covering and bottom-up combination to form a hierarchy of use cases.

Granular relationships: *inheritance* (specialization and generalization) in which a large case is decomposed into small cases or a set of small cases are combined to constitute a large case; *inclusion* where the source case has some of its steps provided by the target case; and *extension* where the source case adds steps to the target case.

Granular computation: identifying the process of use cases, including input data, output data and business logic, as well as the communication with actors.

Consider the following example of the user requirement from a car rental company [34]:

Since we automated the tracking of cars at our stores – using bar codes, counter-top terminals and laser readers – we have seen many benefits: the productivity of our rental assistants has increased 20%, cars rarely go missing and our customer base has grown strongly (according to our market research, this is at least partly due to the improved perception of professionalism and efficiency).

The management feels that the Internet offers further exciting opportunities for increasing efficiency and reducing costs. For example, rather than printing catalogs of available cars, we could make the catalog available to every Internet surfer for browsing on-line. For privileged customers, we could provide extra services, such as reservations, at the click of a button. Our target saving in this area is a reduction of 15% in the cost of running each store.

Within two years, using the full power of e-commerce, we aim to offer all of our services via a Web browser, with delivery and pick-up at the customers’ home, thus achieving our ultimate goal of the virtual rental company, with minimal running costs relative to walk-in stores.

From above requirement, we can identify some use cases (granules) as follows:

U1: Browse index: A customer browses the index of car models;

U2: View Result: A customer is shown the subset of car models that was retrieved;

U3: View Car Model Details: A customer is shown the details of a retrieved car model, such as description and advert;

U4: Search: A customer searches for car models by specifying categories, makes and engine sizes;

.....

One can find out all the use cases to cover the business requirements, although some of them may overlap. Some relationships among these use cases can be also identified. For example, one can see U1 is a special case of U2 without specifying any search criteria. Thus U1 is a specialization of U4 and U4 is a generalization of U1. Similarly, U2 is included in both U1 and U4, since after browsing (U1) and search (U4), the result must be shown to the customer (U2). On the other hand, U2 is extended by U3 to view the car model details after viewing the search/browsing results. Readers can identify other relationships amongst these use cases.

3.2. System Analysis

The basic goal of system analysis is to find candidate classes that describe the objects that might be relevant to the system, relationships between the classes, as well as attributes for the classes. With granular computing terminology, classes are granules that all together should cover the system requirement and satisfy the needs of use cases that have been identified before.

Granulation method in system analysis is to identify classes. Candidate classes are often indicated by nouns in the use cases except those that represent the system, actors, boundaries, and trivial types. Two basic rules should be followed to identify classes: high cohesion inside classes and low coupling between classes. Class identification might be top-down and/or bottom-up, but the general complete coverage is necessary.

Granular relationships in the system analysis correspond to class relationships, which can include the following types: inheritance (is-a relationship) where a subclass inherits all of the attributes and behavior of its superclass(es); association, where objects of one class are associated with objects of another class; aggregation (strong association), where an instance of one class is made up of instances of another class; composition (strong aggregation) where the composed object can't be shared by other objects and dies with its composer; and others as well.

Computation with granules is two-fold. First, determine the internal structure and behaviors of objects of classes, including the instance fields and

method/constructor prototypes; and second, interface the connections between classes. Associating classes often affects the design of internal structures of classes.

Going back to the example in the previous subsection, one can design the following classes for the car rental online system.

C1: Customer
C2: Member
C3: NonMember
C4: Rental
C5: CarModel
C6: Car
C7: CarDetails
C8: InternetAccount
C9: CreditCard
C10: Address

Some relationships between above classes can be extracted. For example, *A Car can be rented under a Rental; A Member is-a Customer; A NonMember is-a Customer; A Member is guaranteed by a CreditCard; A Car is an example of a CarModel; etc.*

Granular computing strategy has been extensively used in software system analysis, especially in designing classes in object-oriented analysis. The followings are two typical examples in Java-based systems. One is *Collection*, an interface in JDK. The architecture of collections in Java is illustrated as follows [36]:

Collection (Interface)
List (Interface)
ArrayList (Class)
LinkedList (Class)
Set (Interface)
HashSet (Class)
SortedSet (Interface)
TreeSet (Class)

In this architecture, the interface *Collection* has two sub-interfaces: *List* and *Set*. *List* has two implementing classes: *ArrayList* and *LinkedList*, while *Set* has one implementing class *HashSet* and one sub-interface *SortedSet* that is implemented by a class *TreeSet*.

Another typical example in Java is *Exception*, where *Exception* is granulated into *IOException*, *ClassNotFoundException*, etc. some of which are granulated further into sub-exceptions. Part of various exceptions and their (inheritance) relationships is shown as follows [6]:

Exception
IOException
EOFException
FileNotFoundException
.....
ClassNotFoundException

CloneNotSupportedException
RuntimeException
 ArithmeticException
 ClassCastException
 IllegalArgumentException
 NumberFormatException
 IllegalFormatException

 IllegalStateException
 IndexOutOfBoundsException
 ArrayIndexOutOfBoundsException
 NoSuchElementException
 NullPointerException

3.3. System Design

Basically the software system design is to decompose a system into physical and logical components, and determine the technologies to be used to implement the system. Traditional system design focuses on the system functional partitioning, while modern software system design is based on object-oriented technology. In this subsection, object-oriented system architecture design and technology are discussed from the perspective of granular computing.

System design involves multiple steps, including system topology, software partitioning, concurrency and security policies as well as communications between components. Our discussion will concentrate in the topology design and system partitioning.

The first task of the system design is normally to determine the topology of a networked system. One popular system topology is the three-tier architecture to separate user interfaces, program logic and data in the system. In this architecture, the client tier presents the user interface to the user so that he/she can enter data and view results; the middle tier – also known as the business logic tier or server tier – runs multi-thread program code using large processors and lots of memory; and the data tier stores the data and provides safe concurrent access to it, typically with the help of a database management system. Thus, the system is granulated into three components. Besides the design of these three tiers, the protocols between tiers are also important. Existing technologies can be chosen to implement each of them, and shown below [8].

Client tier technologies in Web-based systems
 HTML forms
 JavaScript
 ActiveX controls
 Java applets
Client tier to middle tier protocols
 IMAP (email)

HTTP/CGI
FTP / Telnet
TCP/IP
JRMP / IIOP
Middle tier technologies in Web- based systems
 JSP for building Web pages on-the-fly
 ASP for Microsoft technology
 CGI scripts written in languages/ PERL
 Servlets accessible by Java applets or JSPs
Middle tier to data tier protocols
 JDBC net with Java-based system
 EJBs
 JRMP
 TCP/IP
 IIOP
Data tier technologies
 DBMS
 JDBC net server
 Database client
 EJB data source

To granulate the software, thus, object-oriented system design partitions the system components into layers as below:

User Interface / Swing
Control
Network / RMI
Server
Business
Persistence
Database / JDBC

3.4. System Implementation

Granular computing strategies have also been broadly used in software system implementation. The typical object-oriented program structure is actually a granular structure. Consider a Java-based software system, which can be characterized as follows:

Software system
 Packages
 Classes
 Instance fields
 Constructors
 Methods
 Interfaces
 Method prototypes

In this structure, a software system is designed as a set of packages, which may be installed in different physical devices. Each package contains a set of interfaces and a set of classes which are partitioned into three types of components.

During the design of classes, algorithms are one of designer's main concerns. Granular computing is also

an important strategy for this purpose. Let's consider divide-and-conquer technique that is frequently used in algorithm design. The general paradigm of divide-and-conquer is [37]

Divide: divide the problem S in two or more disjoint subsets S_1, S_2, \dots

Recur: solve the subproblems recursively

Conquer: combine the solutions for S_1, S_2, \dots , into a solution for S

The base cases for the recursion are subproblems of constant size

In the above paradigm, each subproblem is a granule, which can be granulated further into smaller subproblems. Granulation process usually follows the top-down method and the obtained granules should completely cover the parent problem. The granularity constitutes a hierarchy, where the top is the problem originally given, while at the bottom are all base cases. The computing with granules consists of not only dividing of a non-base problem or solving a base problem but also the combining solutions to subproblems to form a solution to the parent problem.

Another important general algorithm design paradigm is dynamic programming, which is actually a special case of divide-and-conquer, and thus a granular computing strategy. The main difference between them is: divide-and-conquer solves each subproblems individually, while dynamic programming stores solutions to all subproblems so that when a subproblem is reencountered, the solution can be obtained directly without resolving it.

4. Concluding Remarks

Software engineering is both a thinking process and a problem solving process. We examined different phases of object-oriented software development, including requirement analysis, system analysis, system design, and system implementation from the perspectives of granular computing. It evidences that granular computing as a human thinking model plays an essential role in software engineering, although software analyst, designer and developer have been unconsciously applying this strategy in their daily work. We conclude that consciously using granular computing methodology in software engineering will improve the quality of software products.

References

- [1] Ahl, V. and Allen, T. Hierarchy Theory, a Vision, Vocabulary and Epistemology, Columbia Univ. Press, 1996
- [2] Bargiela, A. and Pedrycz W. Granular Computing: an Introduction, Kluwer Academic Publishers, Boston, 2002.
- [3] Giunchiglia, F. and Walsh, T., A theory of abstraction, Artificial Intelligence 56: 323-390, 1992
- [4] Goodrich, M. and Tamassia, R., Algorithm Design: Foundations, Analysis, and Internet Examples, 2nd ed., John Wiley & Sons, Inc. 2004.
- [5] Hobbs, J., Granularity, Proc. of IJCAI, 432-435, 1985.
- [6] Horstmann, C., Big Java, 3rd ed, John Wiley & Sons, Inc., 2007.
- [7] Lin, T.Y., Granular computing, announcement of the BISC Special Interest Group on Granular Computing, 1997.
- [8] O'Docherty, M., Object-Oriented Analysis & Design, John Wiley & Sons, Ltd., 2005.
- [9] Pawlak, Z., Rough Sets, International Journal of Computer and Information Sciences 11: 341-356, 1982.
- [10] Pawlak, Z., Granularity of knowledge, indiscernibility and rough sets, Proceedings of IEEE International Conference on Fuzzy Systems, 106-110, 1998.
- [11] Watt, D., Brown, D., Java Collections, John Wiley & Sons, Inc., 2001.
- [12] Wu, C., and Yang, X., Information Granules in General and Complete Coverings, Proc. of IEEE GrC, 675-678, 2005.
- [13] Yao, J. T., (2006) Information Granulation and Granular Relationships, Proc. of IEEE GrC, 326-329, 2006.
- [14] Yao, Y.Y., A partition model of granular computing, LNCS Transactions on Rough Sets 1: 232-253, 2004.
- [15] Yao, Y. Y. (2005) Perspectives of Granular Computing, Proc. Of IEEE GrC, 85-90, 2005.
- [16] Yao, Y. Y. (2007), Structured Writing with Granular Computing Strategies, accepted to Proc. of IEEE GrC 2007.
- [17] Zadeh, L. A., Fuzzy sets and information granularity, Advances in Fuzzy Set Theory and Applications, M Gupta, R. Ragade and R. Yager (eds.), North-Holland Publishing Co., 3-18, 1979.
- [18] Zadeh, L.A., Towards a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic, Fuzzy Sets and Systems 19: 111-127, 1997.