A Survey of Business Component Identification Methods and Related Techniques

Zhongjie Wang, Xiaofei Xu, and Dechen Zhan

Abstract**—**With deep development of software reuse, component-related technologies have been widely applied in the development of large-scale complex applications. Component identification (CI) is one of the primary research problems in software reuse, by analyzing domain business models to get a set of business components with high reuse value and good reuse performance to support effective reuse. Based on the concept and classification of CI, its technical stack is briefly discussed from four views, i.e., form of input business models, identification goals, identification strategies, and identification process. Then various CI methods presented in literatures are classified into four types, i.e., domain analysis based methods, cohesion-coupling based clustering methods, CRUD matrix based methods, and other methods, with the comparisons between these methods for their advantages and disadvantages. Additionally, some insufficiencies of study on CI are discussed, and the causes are explained subsequently. Finally, it is concluded with some significantly promising tendency about research on this problem.

Keywords**—**Business component, component granularity, component identification, reuse performance.

I. Introduction

COMPONENT identification (CI) is an important problem in reuse-based software engineering research, and is also considered as a pivotal technique to realize *software ruse* [1]. This is because that component in various granularity levels are the basic unit for composing software systems [2], and only when there exists a set of components that are worthy to be reused, can deep software reuse be really realized [3].

Components can not be obtained baselessly, and designers should follow specific *principles and goals* to analysis some source information (e.g., domain business models) with domain knowledge to find reusable components with good performance. Therefore CI can be defined as *the process of identifying a set of components that satisfy specific performance metrics following some guidelines* [1].

According to different existence forms, components can be classified into two types: *business components* (BC) [4][5] and *software components* (SC) [2][6], the former of which describe business functions or business objects related to reality world [8], represent real-world semantics, but are not concerned about real implementation, and can be regarded as logic components, and can further be classified into *entity-centric* BC and *process-centric* BC [7]; the latter of which are BCs' reflection in software world, and are usually represented as the form of binary codes, and can be regarded as executable components. Regardless of what kinds of components, they both express specific semantics and can be described by component models, e.g., 3C [9], JBCOM [10] for BC, and DCOM, CORBA CCM, EJB for SC.

According to different information source and type of objective components, CI is classified into *forward identification* (FI) and *reverse identification* (RI). FI refers that, in the situation that the objective software system does not exist, designers start from requirement models to identify BCs and implement these BCs as SCs, then use these SCs to construct objective software systems [1][3][7]. FI is an important phase of Reuse-Based Software Engineering (RBSE) [6], Component-Based Development (CBD) [11] and Model-Driven Architecture (MDA) [12]. RI refers that in the situation that software systems have existed, reversely analysis source codes of these legacy systems to identify SCs [13][14][15]. RI has great significance in Reverse Engineering (RE), Program Comprehension, Program Recovery, etc. In conclusion, the process of CI is shown in Fig. 1.

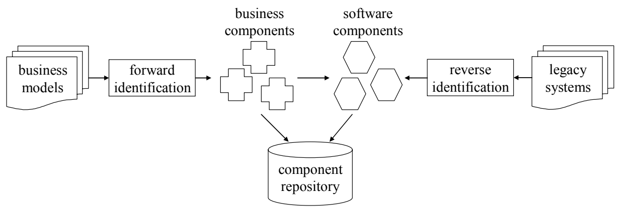

Fig. 1 Basic process for component identification

No matter FI or RI, because of the complexity of source information (i.e., business models, source codes), it is not advisable for component designers to identify components manually. With the aid of proper algorithms or automatic tools, efficient CI activity can be accomplished agilely.

In fact, reverse engineering (RE) was paid attention to much earlier than forward engineering (FE). Since 1990s, with the popularization of CBSE [6] and wide application of a large number of complex Enterprise Software and Applications (ESA), e.g., ERP, CRM, etc, how to rapidly and effectively identify reusable components from domain models has become

Manuscript received September 2, 2005. This work was supported in part by the Specialized Research Fund for the Doctoral Program of Higher Education (SRFDP) in China (Grant No. 20030213027) and the National Natural Science Foundation in China (No. 60573086).

Zhongjie Wang, Xiaofei Xu and Dechen Zhan are with the Research Center of Intelligent Computing of Enterprises (ICE), Harbin Institute of Technology (HIT), 150001 Harbin, China (phone: 86-451-86412664; fax: 86-451-86413309; e-mail: {rainy,xiaofei,dechen}@hit.edu.cn).

more and more urgent, so forward engineering has been gradually emphasized by researchers and practitioners. Many methods in RE were used for reference in FE, and there have appeared many new CI methods. In this paper, we mainly summarize some widely accepted methods for CI in forward engineering, and try to provide up-to-date research process in this field for related researchers.

The rest of this paper is organized as follows. In section 2, we briefly introduce some basic background and technical stake of CI, including identification goals, input model form, identification strategies, identification process, etc. In section 3, typical CI methods in literatures are classified into four types, i.e., domain engineering based methods, clustering based methods, CRUD based methods and other methods, and we address characteristics of each type of methods. In section 4, qualitative comparisons between four types of methods are put forward. Additionally, in section 5 some disadvantages of study on CI are discussed and the causes are explained at the same time. Then it is concluded with some significantly promising tendency about research on CI. Finally is the conclusion in section 6.

II. Technical Stake for Business CI

CI is a main task in *domain engineering*, in which a set of reusable components are obtained by analysis, clustering and abstraction on domain business models. Inversely, by reusing these components, concrete business models and the corresponding software systems are implemented. There is a strict and bi-directional mapping between business models and business components, i.e., components are software representation of business models, and business models are the semantics representation of components.

Actually, starting from domain models and clustering them to obtain reusable components, is consistent with basic ideas of MDA [12]. MDA is a research hotspot in software engineering in recent years, which emphasizes automatic mappings between models in different levels, e.g., CIM, PIM, PSM and codes [16]. Domain business models represent universal and common business requirements in specific domains, and are considered as CIM, and business component models belong to PIM, therefore CI can be considered as the transformation from CIM to PIM.

Related aspects in CI include identification goals, input model forms, identification strategies and process, etc. Differences between these aspects lead to different identification methods.

A. *Identification Goals*

To cluster business models into components is not at will, or random, and some definite principles must be followed to ensure final components have good performance and high reuse value.

Initial component design principles mostly originates from design principles of *Class* and *Package* in object-oriented (OO) methods[8], such as Open-Close Principle (OCP), Dependency Inverse Principle (DIP), Interface Separation Principle (ISP), Release Equivalent Principle (REP), Common Reuse Principle (CRP), Common Close Principle (CCP), Single Responsibility Principle (SRP), Acyclic Dependency Principle (ADP), Stability Dependency Principle (SDP), Stability Abstraction Principle (SAP), etc [17]. These principles normalize class design from semantics and structural aspects, and since there are some similarities between component and class in some aspects, they are imported into component design.

However, a component is not the simple aggregation of classes, and there exists essential difference between them, therefore, these principles are not fully suitable for component design. Even if a component is considered as the aggregation of static classes [18], these principles are limited to the design of entity components, and it is difficult for them to instruct design of process components.

With the deep research on CBSE, delegated by some new methodologies, e.g., Business Component [4], Catalysis [19], UML component [20], some basic component design principles and methods was presented from the view of whole component development process, but have not obtained enough attentions. In addition, with the gradual improvement of component reusability evaluation methods, various performance metrics are put forward and widely applied in practice, such as the five management metrics (cost effectiveness, ease of assembly, customization, reusability, maintainability) and five technical metrics (coupling, cohesion, number of components, size of component, complexity) in [21]. By summary on related researches, we think that CI should pay more attention on those quantitative metrics, as shown in Table I, by balancing between these metrics to realize global optimization on component performance.

TABLE I
OPTIMIZATION GOALS FOR CI

Metrics		Symbol		Meanings	Influence factors	Optimization
Reusability[6][21]		*R* (*C*)		The scope that *C* could be reused in, or the frequency that *C* could be often reused	Semantics commonality and variability of functions contained in *C*	Maximum
Reuse cost [6][21]	Instantiation cost	*RC*(*c*)	*IC*(*C*)	The cost to eliminate variation points in *C* for a specific requirement	Number of variation points in *C*	Minimum
	Implementation cost		*PC*(*C*)	The cost to implement those unimplemented extended points in *C* for a specific requirement	Number of unimplemented extended points in *C*	Minimum
	Composition		*CC*(*C*)	The cost that C composites with other components	Number of interfaces and	Minimum

	cost			by interfaces to form integrated software systems	complexity of parameters/data in interfaces	
	Change cost		*GC*(*C*)	The cost to reconfigure/modify *C*’s structure and functions to fit for unsupported requirements	Stability of *C* and complexity of functions in *C*	Minimum
Reuse efficiency[6]		*RE* (*C*)		The contributions that *C* has to construct software systems	Granularity	Maximum
Stability[22]		*S* (*C*)		The degree that functions in *C* need frequently change	—	Maximum
Granularity[2][7][21]		*G* (*C*)		The scale of *C*, or the number of functions contained in *C*	—	Maximum
Cohesion[3][7][21]		*Cohesion* (*C*)		Semantics closeness between functions in *C*	—	Maximum
Coupling[3][7][21]		*Coupling* (*C*)		Semantics closeness between functions in *C* and in other components	—	Minimum

The above metrics *restrains mutually*, and cannot reach optimization at the same time [6]. For example, coarse-grained components have higher reuse efficiency but lower reusability, and vice versa. Another example is that, components with higher reusability are sure to have higher instantiation and implementation cost. Therefore, CI is a *multi-objective optimization problem*, i.e., under the guarantee that the reusability, reuse efficiency, cohesion, granularity, stability are as high as possible, to ensure reuse cost and coupling as low as possible.

Suppose we have identified *n* components $\{C_1, C_2, \ldots, C_n\}$ from a business model *BM* and form component set *C_set*, we can use the average of each metrics as the objective function of optimization, i.e.,

$$\frac{1}{n}\max\left\{\sum_{i=1}^{n} R(C_i)\right\}, \frac{1}{n}\max\left\{\sum_{i=1}^{n} RE(C_i)\right\}, \frac{1}{n}\max\left\{\sum_{i=1}^{n} G(C_i)\right\},$$

$$\frac{1}{n}\max\left\{\sum_{i=1}^{n} S(C_i)\right\}, \frac{1}{n}\max\left\{\sum_{i=1}^{n} Cohesion(C_i)\right\},$$

$$\frac{1}{n}\min\left\{\sum_{i=1}^{n} Coupling(C_i)\right\},$$

$$\frac{1}{n}\min\left\{\sum_{i=1}^{n}\left(IC(C_i)+PC(C_i)+CC(C_i)+GC(C_i)\right)\right\}.$$

Then the integrated objective function of CI can be denoted as:

$$\max Z(C\_set),$$

$$Z(C\_set)=\frac{\sum_{i=1}^{n} R(C_i)\times\sum_{i=1}^{n} RE(C_i)\times\sum_{i=1}^{n} Cohesion(C_i)\times\sum_{i=1}^{n} G(C_i)\times\sum_{i=1}^{n} S(C_i)}{\sum_{i=1}^{n} RC(C_i)\times\sum_{i=1}^{n} Coupling(C_i)}$$

In these metrics, *granularity* is one of the most important one, whose influence on component performance has been gently noticed by researchers. With the incessant update of software reuse techniques, reusable artifacts have developed from initial functions, objects [2], to components, frameworks, design patterns [6], until today, to software architecture and web services in Internet-based environment. It is easy to see that granularities of reusable artifacts are changing from fine-grained to coarse-grained step by step [23], and coarse-grained reuse has become a major tendency of software reuse.

In coarse-grained CI, we can obtain coarse-grained entity components by assembling fine-grained entity components together according to the generalization and composition relationships between them [18]. But for coarse-grained process components, at present there are still no proper methods in literatures yet.

Although current trend are coarse-grained reuse, component granularity is not “the coarser, the better”. In a software organization, the granularity level of accumulated and reused components is usually determined as a decision before reuse projects start [6]. In the chosen granularity level, each concrete component’s granularity is usually determined by designers’ experiences during CI and design phase. Therefore, how to appoint proper granularity to each component, to make granularity as coarsest as possible under the premise of avoiding deficiencies brought by coarse-grained granularity, is an important goal of CI.

B. Form of Input Models

The inputs of CI are business models. According to different modeling tools, these models can have different forms. In view of the fact that UML has become the standard of software modeling, most of CI methods adopt *UML models* as the way to express business semantics [1][3][7][24][25][26], such as:

- UML use case diagram
- UML class diagram
- UML process diagram
- UML collaboration/sequence diagram

UML models contain business elements in software model levels, e.g., objects, operations, events, which are all finer-grained semantics, and lacks of the ability to support coarse-grained semantics modeling, therefore, besides UML models, other forms of models are also adopted, e.g., domain feature models [27][28], business goal decomposition models [8], etc.

Models in different forms may be represented as a uniform form using *feature space* as a tool [27], i.e., transforming the models into a feature tree, in which features with the same semantics types (e.g., business process, business activities, business operations, business objects, etc) are in the same layer, and there exists composition/aggregation relationships between features in neighboring layers, and dependency or association relationships between features in the same layer. Generalization and specialization relationships can be expressed by feature’s “Type-Value” mechanism [28].

Most models are expressed in graphical forms, which algorithms cannot directly deal with, therefore, before CI process, these graphical models should be pre-treated, i.e., analyzing semantics dependencies and the corresponding dependency intensity between elements in models and transform the model into the form of matrix or weighted directional graph to be the input of automatic identification algorithm.

C. *Identification Strategies*

Business components provide specific services to outside via interface, and these services could support implementation of one or several business models in enterprises, therefore, a component can be regarded as a partial model of a global business model [28], i.e., the sub feature space of business models' feature space. There exists a mapping between them, and how to create this mapping between business model space and component space and decompose business model space into a set of components according to this mapping, is the core problem in CI.

This mapping should satisfy characteristics of *completeness* and *non-intersection*. *Completeness* refers that any business element e contained in business models can be implemented by the composition of n components. n=1 means e can be implemented by features of one specific component, and n>1 means the element has to be implemented by features from all components in $\{c_1,c_2,\ldots,c_n\}$ by composition. *Non-intersection* refers that for arbitrary two components, their feature spaces are not intersected, i.e., one business element is not allowed to appear in two components simultaneously to avoid redundancy or inconsistency.

There are two general approaches to partition a given domain feature space into component form [4]. First there is what might be called *continuous recursion*, which is an analysis technique in which the problem space is partitioned by identifying very coarse grained components, then each very coarse grained component is partitioned into components of a lower granularity, and so on iteratively until the desired granularity is achieved. The second approach can be termed discrete recursion. While supporting strongly the concept that components are made up from smaller components, discrete recursion defines specific granularity levels. Each level identifies a unique category of component. Each category has specific characteristics that address the requirements for that level, including defined relationships with the other categories.

Refined further, this mapping is classified into four types: *single granularity level mapping* (SG), *multiple granularity level mapping* (MG), *middle granularity level mapping* (IG) and *dynamic granularity level mapping* (DG). In these mechanisms, domain business models are firstly transformed into the form of feature space, which are then partitioned into a set of sub space, each of which is mapped into a component.

In SG, firstly a specific granularity level is chosen, and each element in this level with its all descendant elements is directly mapped to a component; for each element above this level, it can be implemented by composition of components just obtained.

MG is an extend of SG mapping, i.e., several granularity levels are selected at one time, then do SG for each selected level and obtain components in these chosen granularity levels.

A common deficiency in SG and MG is that, the final components' granularities are fixed, i.e., when a specific level is chosen, the final components' granularity is equivalent to the chosen granularity level, and cannot change.

The basic idea of IG is: choose one granularity level, then according to some specific coupling relationships, cluster business elements in this level, and each cluster is mapped to a component. For example, if business activity level is chosen for IG, then we can get a set of components, each of which is composed with one or several business activities.

The granularity of components by SG, MG and IG are basically decided when the granularity level(s) is (are) chosen. More importantly, they all do not consider the semantics characteristics of business elements themselves, which makes component granularity completely not related with business elements.

The last strategy is DG to realize fully dynamic granularity. If we integrate SG, MG and IG together, for arbitrary one business element e in arbitrary levels of business model space, there may be three possible strategies for e to be mapped to component space:

- Directly mapped as a component c, i.e.,
$$\Omega(c)=\{e\}\cup descendant(e)$$
- Mapped as part of a component c, i.e.,
$$\Omega(c)\supset\{e\}\cup descendant(e)$$
- Mapped as composition of several components $\{c_1,c_2,\ldots, c_n\}$, i.e., $\bigcup_{i=1}^{n}\Omega(c_i)\supseteq\{e\}\cup descendant(e)$

The key of DG is the mapping principles, i.e., according to which of the above three strategies a business element is mapped to component space.

In Table II we briefly summary the difference between four mapping strategies.

TABLE II
COMPARISONS BETWEEN FOUR MAPPING STRATEGIES

	SG	MG	IG	DG
Fixed level	Yes	Yes	Yes	No
Number of mapping levels	1	*n*	1	*n*
Mapping strategies	Static	Static	Dynamic	Dynamic
Component granularity	Static	Static	Dynamic	Dynamic

D. *Identification Process*

Mapping between business space and component space can be denoted by the following equations:

$$C\_set\equiv Aggregate\Big(Abstract\Big(Decomposite\big(BM\_set\big)\Big)\Big) \quad (1)$$

$$BM\equiv Composite\Big(Config/Instantiate/Adapt\Big(Select\big(C\_set\big)\Big)\Big) \quad (2)$$

Equation (1) refers to CI process, by decomposition and abstraction on a set of business models *BM_set* to get a set of components *C_set*. This process contains five basic sub-phases:

- Partitioning: Cluster business models into sub models according to specific principles, and map each sub model into a self-contained component. There are many partition principles, such as cohesion-coupling based decomposition [1][3][7], data dependency and function dependency based decomposition [6], etc.

● Abstraction: Abstract those similar services in different components into an abstract service so that components can be applied in multiple business situations to increase reusability. This process can also be called "*variation point design*" [6][40], with some example techniques are: dimension reduction, grouping, splitting, and intensionalization [6], to replace low-order variabilities by a higher-order commonality.

An abstract component might implement multiple variabilities of a specific business, i.e., different implementations of the business, which can be called "*vertical abstraction*". This kind of components usually deals with the same business objects (data) with different business logic (rules). For example, in a component "purchasing product arrival process component", there will be different business logic due to the different arrival order of products and purchasing invoice. An abstract component can also realize a common sub function in multiple businesses, and is called "*horizontal abstraction*". This kind of components has most common and a few special business logics (rules), and can support to deal with different business objects (data). For example, component "sale order management" can deal with multiple order types, e.g., ordinary orders, retail orders, long-term orders, etc.

● Aggregation and decomposition. To realize optimization on performance, for those components that are often reused together, aggregate them into a single coarser-grained component to increase reuse efficiency and decrease reuse cost. Related techniques include Common Reuse Principle (CRP) [17], generalization/composition based aggregation [18], etc. Contrariwise, granularity can also be decreased by decomposing one coarser-grained component into several finer-grained ones.

● Structure design: for each business component, design its inner functional structure, outer interfaces, and relationships with other components [7][24][25].

● Performance evaluation: for final component sets, choose specific performance metrics and evaluate them [1][3][7]. If the evaluation results do not satisfy expectations, then turn into the identification process again to re-identification or re-design these components.

In Fig. 2, we present the detailed process of CI.

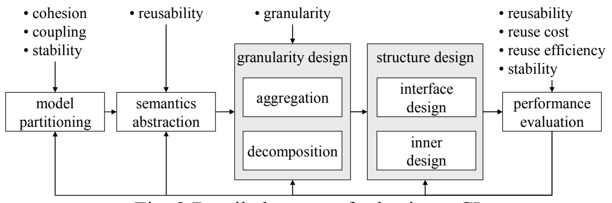

Fig. 2 Detailed process for business CI

Equation (2) is the reverse process of (1), denoting the process of reusing components to construct software systems.

III. Classifications on Business CI Methods

Aiming at different business model forms, identification goals and strategies, researchers have presented various identification methods, and formed comparatively mature CI methodologies, which can be classified into four types: Domain Engineering (DE) based methods, Clustering Analysis (CA) based methods, CRUD matrix (CM) based methods, and other methods.

A. Domain Engineering based Methods

As mentioned above, initially CI has been considered as a phase in domain engineering [6], in which component designers do domain analysis from a group of similar requirements in one business domain, find commonalities and variabilities across them, construct domain specific software architecture (DSSA) to seek reusable business semantics, then construct reusable business component specifications.

Researches on domain engineering started from early 1980s, and by 20 years' development, at present typical and popular research and practice work include: Feature-Oriented Domain Analysis (FODA) [29], Feature-Oriented Reuse Method (FORM) [30], Product Line Method (PLM) [31], Reuse-Driven Software Engineering Business (RESE) [5], JadeBird Object-Oriented Domain Engineering [32], etc. These methods usually focus on the reusability of domain architecture and adaptability of objective components, whereas does not quite emphasize on performance factors, e.g., reuse cost and reuse efficiency. In addition, these methods rarely have the ability to obtain reusable components from business models automatically, and CI should be accomplished with the aid of experiences of domain analyzers.

Up to now, these methods have not taken the concept "stability" into consideration yet, but emphasized on analysis of commonalities and variabilities (C&V) with a basic hypothesis that, the commonalities in domain is always stable. Actually it is not reasonable. This is because any software artifacts require changing itself along with time [33]. Hamza and Fayad did some research on stability of software systems [22], and present Stability-Oriented Domain Analysis (SODA) [33] method, in which commonalities are classified into enduring and instable ones, accordingly software is partitioned into three layers: Enduring Business Themes (EBTs), Business Objects (BOs) and Industrial Objects (IOs), to realize clear separation of stability. But SODA does not produce constructive opinions on the optimization of component granularity, either.

B. Cohesion-Coupling based Clustering Analysis Methods

In afterwards research from 1990s, CI has been regarded as an independent problem and obtained widespread attentions. Starting from considering reuse cost optimization, researchers try to cluster business models according to "high cohesion and low coupling" principle and encapsulate each cluster into a component [3]. Basic ideas of these methods are: calculate the strength of semantics dependencies between two business elements and transform business models into the form of weighted directional graph, in which business elements are nodes and semantics dependency strength are the weight of edges between two nodes, then cluster the graph using graph clustering or matrix analysis techniques. This type of methods is summarized in [34].

Clustering analysis is a method in mathematical statistics for precise classification. It aggregates those elements with high cohesion together to form specific patterns, and is widely used in the field of data mining and pattern recognition. Researchers imported it into CI and expect to obtain components with high cohesion and low coupling to reduce composition cost. Depending on different strategies of calculating dependency strength (DS) between nodes, clustering analysis may produce different results. The basic process is presented in [7], just as follows:

(1) Denote *n* elements that need to be classified as set *X*, and initially each element in *X* forms a cluster;

(2) Specify the principles for calculating DS, i.e., similarity between arbitrary two nodes, and denote DS between X_i and X_j as R_{ij};

(3) Calculate DS between arbitrary two nodes in *X* and obtain the DS matrix *D* of *n* elements;

(4) Choose a sound "Minimum DS" R_{min} as the judgment principle for merge two elements into one cluster;

(5) According to each R_{ij} in *D*, execute the following clustering process:

(5.1) (Valve value) If $R_{ij} \geq R_{min}$, then set X_i and X_j into one cluster;

(5.2) (Transitivity) If X_i and X_j, X_i and X_k belong to the same cluster respectively, then merge X_i, X_j, X_k into one cluster;

(6) Map elements in each cluster together into a business component.

In this process, key techniques need to be concerned include: how to calculate SD between nodes, how to cluster the graph. Aiming at the former, static SD and dynamic SD are separately calculated and then combined together to get the final SD [1][3][7]. Static SD is mainly resulted by relationships between business objects, e.g., *generalization*, *composition*, *aggregation*, etc; and dynamic SD is mainly resulted by relationships between use cases or business objects, e.g., *use* and *function call*, etc. By set a specific weight for each type of dependency and sum up the weights of all dependencies between two nodes, the global SD is obtained. Different clustering methods have different but similar calculation method.

Aiming at the latter, graph clustering or matrix analysis is usually adopted. Identifying sub-graphs with high cohesion is considered as an equivalent problem of identifying strongly connected sub-graphs [35]. Graph clustering is a classic research problem in graph theory, and there are many clustering algorithms in literatures, such as *k-cut* based clustering [36], maximum flow and minimum cut network clustering algorithm [37], etc. Since graph clustering is an NP-hard problem [38], and cannot get optimal result in polynomial-time, some heuristic algorithms, e.g., *genetic algorithm* [38][39], are usually adopted to obtain approximate optimal results.

However, heuristic algorithms still need a large number of iterations, and in order to improve efficiency, some *approximate algorithms*, e.g., top-down or bottom-up hierarchical algorithm [13], Chameleon algorithm [40], Core Entity algorithm [7], etc, to identify components for less execution time and acceptable results.

Typical methods in this type include:

In [3], a basic cluster algorithm was presented, in which business entities was taken as nodes, and the strength of relationships between entities was taken as weight. If the weight between two nodes is higher than the per-set valve value, then they are in the same cluster. By property of transitivity, the final clustering is obtained.

In [7], a Cluster Algorithm is adopted to identify two kinds of business components (process component and entity component) and requirement models are taken as the data source of Cluster Algorithm samples. Several formulations are given to calculate the value of samples' relationship. Based on [3], Core Entity was chosen to achieve better accuracy of Business CI. Several selection strategies of core entity were presented, and several peculiar situations are also taken into considerations.

In [1], a heuristic algorithm is adopted for clustering. It starts from object models in analysis model level, uses hierarchical clustering method to get initial clustering scenario, then applies a set of pre-defined constraints and heuristic rules, e.g., move/exchange objects between different clusters, add new clusters, etc, to get a new cluster scenario, which will be evaluated carefully to determine whether it may be accepted as the final results. When the iteration process stops, an approximately optimal cluster scenario is obtained.

In [35], a coupling analysis method to identify business components is presented. It aims at business process models mainly and considers three connection relationships (serial, parallel, and coupling) between processes to identify sub-processes with high coupling and set them in a process component. This method uses graph adjacency matrix as a tool and by matrix transformation and block to cluster models.

Components identified in this type of methods are loose coupling and high cohesion both in semantics and structure, so as to ensure lower reuse cost. These methods also support to cluster business models into components automatically, but they only aim at specific business model types, and have not considered reusability and adaptability of components. In addition, they use IG mapping strategy, which cannot realize optimization on granularity, i.e., the final components' granularities are relatively fixed in one or several levels. What is more, they try to pursue balance between granularities of different components sedulously, or even decrease granularity by decomposition [7]. This runs in the opposite direction of coarse-grained reuse tendency.

C. CRUD based Methods

Most of clustering based CI methods try to optimize component performance from the view of *coupling-cohesion*, which not only ignores other performance metrics, but also ignores semantics of business elements themselves, therefore the semantics integrity of final components cannot be ensured completely. Researchers have reached a consensus that simply seeking optimal solutions is not quite significant. Therefore, *CRUD matrix* based CI methods [4][24][25][26] appeared.

This type of methods is actually also a clustering method, which uses those behavioral business elements (e.g., use case [24], events [25], operations) and static business elements (e.g., business entities) as sample data, uses four semantic relationships (Create-C, Read-R, Update-U, Delete-D, with the priorities as C>D>U>R) between behavioral and static elements to calculate association weight, and merges those use cases and entities with C or D relationships into one business components.

Fig. 3 shows an example of CRUD matrix.

Event, Method, Use Case, etc	Entity, Object, etc				
		E_1	E_2	E_3	E_4
	M_1	C	C		R
	M_2	U	U	U	R
	M_3	R	R	RU	
	M_4				RUD

Fig. 3 An example of CRUD matrix

Lee and Yang presented a UML model based *Object-Oriented Component Development Methodology* (COMO) [24], in which by analysis on use case diagrams, class diagrams and sequence diagrams, "use case/class matrix" are created. Then "use case and class clustering algorithm" are applied to the matrix to partition it into blocks, accordingly those use cases and classes in the same block are with tight cohesion and aggregated into one business component.

O2BC (*Objects to Business Components*) method [25] presented by Ganesan and Sengupta also bases on UML models, from which Domain Object Model (DOM) and Entity-Event Interaction Matrix are constructed. By several transformation rules the matrix is clustered to blocks to get final entity and process components.

In [26], during the development process of component-based web applications, CRUD matrix between business activities and business objects are adopted to allocate objects into business activity components. Abstraction mechanism is also imported to form concrete business components for specific businesses and common business components for multiple businesses.

This type of methods fully considers semantics relationships between business elements (denoted as C/R/U/D) so that transaction and semantics integrity can be ensured. Its shortcoming is that other performance metrics are not addressed, either.

D. *Other Methods*

Besides three types of CI methods above, there are still some other methods, but they have not yet form mature technique system, therefore not the mainstream of CI methods.

Business goal decomposition oriented CI method [8]. This method does not use UML models as input, but uses enterprise business processes, business rule models, etc, to construct business goal space, represented as the form of *Goal Service Graph* (GSG). By decomposing GSG, final components could encapsulate rich design decision information, therefore tight traceability between enterprise businesses and component models is ensured.

Other methods include: Similarity-based CI method [34], Variation Oriented Decomposition (*VOD*) method [41], Information Loss Minimization based method [42], Business Model Stability based method (STCIM) [43] , etc. We will not discuss them in details.

IV. Comparison and Analysis between CI Methods

The four types of CI methods focus on different aspects, so the performances of final components also have big diversity between them. We summarize these differences in Table III.

TABLE III
Comparisons between Different CI Methods

		DE-based methods	CC-based Cluster methods	CRUD-based methods	Other methods
Model forms		Domain feature models	UML use case diagram UML class diagram UML activity diagram UML sequence diagram etc	UML use case diagram UML class diagram External business events Business activity models	Goal decomposition models, Business process models, etc
Application domain		Not limited	Entity components Process components	Entity components Process components	Mainly process components
Goals	Reusability	High	N/A	N/A	High
	Reuse cost	N/A	Low	Low	High
	Reuse efficiency	N/A	N/A	N/A	Low
	Stability	Clear separation	N/A	N/A	N/A
	Granularity	N/A	Seek balance between different components. More attentions on business object level, so granularities are usually fine	N/A	Seek coarse-grained components
	Cohesion	N/A	High	High	N/A
	Coupling	N/A	Low	Low	N/A
Phases	Clustering	√	√	√	√

	Abstraction	√	×	×	√
	Merge/Decomposition	√	√	×	×
	Interface design	×	√	√	√
	Performance evaluation	√	√	×	√
Identification strategies		SG/MG	IG	IG	IG
Tool support		No algorithms, with CASE tool support	With algorithms, but seldom tools	With algorithms, no tools	With algorithms and design process, usually no tools
Typical methods		FODA[29] FORM[30] SODA[33] JadeBird OODA[32]	Cohesion-coupling based methods[1][3] Core entity methods[7]	COMO[24] O2BC[25]	Goal decomposition oriented method[8] VOD[40]

V. Disadvantages and Future Research Work on CI Methods

Although there have been rich research results in CI methods, in practice, CI methodologies are still not quite perfect, and there still lack of standard method architecture and explicit instructions for practical application [7]. Summarizing on current research work in CI methods, we think that there are the following shortcomings:

(1) There lacks of a uniform component semantics model. Although in literatures there are various component models, e.g., 3C, Wright, JBCOM, different CI methods supports different component models and cannot fit for other models, which leads to poor adaptability.

(2) There lack of uniform business models. Similarly, current popular business modeling methods and languages are quite rich, such as UML, UEML, EPC, etc, but different CI methods aim at different business models.

(3) There lacks of a complete component performance evaluation method. Various CI methods only pay attentions to part of performance metrics and ignore others, which lead to incompleteness of component performance, i.e., some performance is quite good, while others are quite bad.

(4) There lacks of tool support. Most of methods have low automation degree, i.e., have to be done manually, and currently there only exist a few tools (e.g., CompMaker [1]) to support automatic CI, and in most situations, it requires component designers to manually identify components.

Aiming at the promising tendency about research on CI methods, we think that future work should be carried out from the following views:

(1) Multi-objective CI methods: integrate those mutually restrained metrics together and try multi-objective optimization in CI process to realize optimal solutions. A feasible plan is to combine current CI methods together, e.g., using DE-based methods for optimization on reusability and stability, using cluster analysis based and CRUD-based methods to optimize reuse cost, and use other methods for optimization on granularity and reuse efficiency, etc.

(2) Dynamic granularity. There are no methods that adopt DG strategy, therefore component granularities are not very flexible and not closely associated with business semantics. Future research should try to set different granularities for different business elements according to semantics characteristics, therefore realize dynamic granularity CI.

(3) Integration of CI and business modeling. Current CI methods usually provide corresponding algorithms and process, but the concrete work has to be done by designers manually. CI and business modeling tools should be integrated together, i.e., embedding CI methods into modeling tools, so as to realize automatic identification after business models are built.

(4) Component reconfiguration. After components are identified and reused in practice for some periods, according to the accumulated reuse data, analyze deficiencies in component design that are not suitable for reuse, then re-identify or re-design these components on structure and semantics to make them more fit for practical reuse.

VI. Conclusions

As a hot research field tending towards mature, research on CI methods connects business models and component models together, and according to specific goals and strategies to create reusable business component and provide valuable assets for software reuse.

At present research on CI is still very active, and it has been considered as an important sub-problem in MDA research. There appears a large quantity of papers annually in some famous international conferences and journals, e.g., ICSE, etc, which proves that this problem is still being widely paid attention to by researchers.

In addition, with the development of web services and the popularization of inter-enterprise software and applications, the problem of web service identification and design has been already underway [44][45] therefore, it is also a research field worthy to be concerned with.

References

[1] H. Jain, N. Chalimeda, N. Ivaturi, and B. Reddy, "Business component identification: A formal approach," in *Proc. of the 5th IEEE Int. Enterprise Distributed Object Computing Conf.* Seattle: IEEE Computer Society Press, 2001, pp.183–187.

[2] C. Szyperski, "*Component software: Beyond Object–Oriented Programming*," Addison-Wesley, 1998.

[3] J.K. Lee, S.J. Jung, and S.D. Kim, "Component Identification Method with Coupling and Cohesion," in *Proc. of 8th Asia–Pacific Software Engineering Conf.* Macau, China, 2001. pp.79–86.

[4] P. Herzum and O. Sims, "*Business Component Factory*," New York: John Wiley&Sons, Inc., 1999. pp. 425–529.

[5] I. Jacobson, M. Griss, and P. Jonsson, "*Software Reuse: Architecture, Process and Organization for Business Success*," Addison–Wesley, 1997.
[6] H. Mili, A. Mili, S. Yacoub, and E. Addy, "*Reuse-Based Software Engineering: Techniques, Organization, and Controls*," New York: John Wiley and Sons Ltd., 2002.
[7] W. Xu, B.L. Yin, and Z.Y. Li, "Research on the business component design of enterprise information system," *J. of Software*, vol.14, no.7, pp.1213–1220, 2003.
[8] L. Keith and A. Ali, "A Goal–driven Approach to Enterprise Component Identification and Specification," *Communications of the ACM*, vol.45, no.10, pp.45–52, 2002.
[9] W. Tracz, "Implementation Working Group Summary," in *Proc. of Reuse in Practice Workshop*, J. Baldo and Jr. Alexandria, Eds., IDA Document D–754, Pittsburgh, PA, pp.10–19.
[10] Q. Wu, J. Chang, H. Mei, and F.Q. Yang, "JBCDL: An Object-Oriented Component Description Language," in *Proc. of the 24th Int. Conf. on Technology of Object–Oriented Languages ASIA*, IEEE Computer Soc. Press, Los Alamitos, CA, 1997, pp.198–205.
[11] I. Crnkovica and M. Larsson, "Challenges of component–based development," *The J. of Systems and Software*, vol.61, no.3, pp.201–212, 2002.
[12] D. S. Frankel, "*Model Driven Architecture: Applying MDA to Enterprise Computing*," Wiley, 2003.
[13] T.A. Wiggerts, "Using Clustering Algorithms in Legacy Systems Remodularization," in *Proc. of 4th Working Conf. on Reverse Engineering*. Washington, DC, USA: IEEE Computer Society, 1997. pp.33–43.
[14] X. Zhou, X.K. Chen, J.S. Sun, and F.Q. Yang, "Software Measurement Based Reusable Component Extraction in Object–oriented System," *ACTA Electronica SINICA*, vol31, no.5, pp.649–653, 2003.
[15] Z.M. Zhang, Y.T. Zhuang, Y.H. Pan, "Object-Oriented Software Reverse Engineering," *J. of Computer Research and Development*, vol.40, no.7, pp.1062–1068, 2003.
[16] A. Kleppe, J. Warmer, and W. Bast, "*MDA Explained: The Model Driven Architecture: Practice and Promise*," Addison-Wesley, 2003.
[17] R.C. Martin, "*Agile software development: principles, patterns, and practices*," New York: Prentice Hall, 2002.
[18] G. Li and M.Z. Jin, "A design method for reusable components," *J. of Computer Research and Development*, vol.37, no.5, pp.609–615, 2000.
[19] D.F. D'Souza and A.C. Wills, "*Objects, Components, and Frameworks with UML: The Catalysis Approach*," Reading: Addison-Wesley Longman, Inc., 1998, pp.505–680.
[20] J. Cheesman and J. Daniels, "*UML Components: A Simple Process for Specifying Component–Based Software*," Boston: Addison-Wesley Longman, Inc., 2000.
[21] P. Vitharana, H. Jain, and F. Zahedi, "Strategy–Based Design of Reusable Business Components," *IEEE Trans. Systems, Man, and Cybernetics – Part C: Applications and Reviews*, vol.34, no.4, pp.460–474, 2004.
[22] M.E. Fayad, "Accomplishing Software Stability," *Communications of the ACM*, vol.45, no.1, pp.111–115, 2002.
[23] D. Helton, "The Impact of Large–Scale Component and Framework Application Development on Business," in *3rd Int. Workshop on Component–Oriented Programming*. pp.163–164, 1998.
[24] S.D. Lee and Y.J. Yang, "COMO: A UML-based component development methodology," in *Proc. of 6th Asia Pacific Software Engineering Conf.* Takamatsu, 1998. pp.54–63.
[25] R. Ganesan and S. Sengupta, "O2BC: A technique for the design of component–based applications," in *Proc. of 39th Int. Conf. and Exhibition on Technology of Object–Oriented Language and Systems*, 2001. pp.46–55.
[26] A. Somjit and B. Dentcho, "Development of industrial information systems on the Web using business components," *Computer in Industry* vol.50, no.2, pp.231–250, 2003.
[27] W. Zhang and H. Mei, "A feature-oriented domain model and its modeling process," *J. of Software*, vol.14, no.8, pp.1345–1356, 2003.
[28] Y. Jia, "*The Evolutionary component–based software reuse approach*," Ph.D. dissertation, Graduation School of Chinese Academy of Sciences, 2002.
[29] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson, "*Feature-Oriented domain analysis (FODA) feasibility study*," Tech. Rep., CMU/SEI-90-TR-21, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, USA, 1990.
[30] K.C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol.5, pp.143–168, 1998.
[31] C. Gary, D. Patrick, K.C. Kang, and S. Thiel, "*Product Line Analysis: A Practical Introduction*," Tech. Rep., CMU/SEI-2001-TR-001, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, USA, 2001.
[32] F.Q. Yang, H. Mei, Q. Wu, and B. Zhul, "An Approach to Software Development based on Reuse of Heterogeneous Components and its Supporting System," *Science in China (E)*, vol.40, no.4, pp.405–413, 1997.
[33] H.S. Hamza, "SODA: A Stability-Oriented Domain Analysis Method," in *Proc. of the 19th Annu. ACM SIGPLAN Conf. on Object-oriented programming systems, languages, and applications*, Vancouver, Canada: ACM Press, 2004, pp.220–221.
[34] K. Rainer, "*Atomic Architectural Component Recovery for Program Understanding and Evolution*," Ph.D. dissertation, Institut für Informatik, Universität Stuttgart, 2000.
[35] X.W. Yuan, Z. Qin, and Z.J. Lu, "Identification method of business component based on coupling analysis," *Control and Decision*, vol.19, no.9, pp.1071–1073, 1077, 2004.
[36] J. Christopher, "Computing Program Modularizations Using the k–Cut Method," in *Proc. of 6th Working Conf. on Reverse Engineering*. Oct.06–08, 1999, Atlanta, Georgia.
[37] T. Konstantinos, "*Maximum Flow Techniques for Network Clustering*," Ph.D. dissertation, Princeton University. 2002.
[38] S. Mancoridis, B.S. Mitchell, C. Rorres, Y. Chen, and E.R. Gansner, "Using Automatic Clustering to Produce High-Level System Organizations of Source Code," in *Proc. of 6th Int. Workshop on Program Comprehension*, 1998.
[39] T.N. Bui and B.R. Moon, "Genetic Algorithm and Graph Partitioning," *IEEE Trans. Computers*, vol.45, no.7, pp.841–855, 1996.
[40] G. Karypis, E.–H. Han, V. Kumar, "Chameleon: Hierarchical Clustering Using Dynamic Modeling," *IEEE Computer*, vol.32, no.8, pp. 68–75, 1999.
[41] A. Ali, Z. Hussein, and A. James, "Externalizing Component Manners to Achieve Greater Maintainability through a Highly Re–configurable Architectural Style," in *Proc. of Int. Conf. on Software Maintenance*. IEEE Computer Society, 2002, pp.628–637.
[42] P. Andritsos and V. Tzerpos, "Software Clustering based on Information Loss Minimization," in *Proc. of 10th Working Conf. on Reverse Engineering*.2003, pp.334–344.
[43] Z.J. Wang, X.F. Xu, D.C. Zhan. "Component Granularity Optimization Design Based on Business Model Stability Evaluation, " *Chinese Journal of Computers*. 2006, 29(2): 239–248.
[44] R. Lee, A. Harikumar, C.C. Chiang, H.S. Yang, H.K. Kim, and B. Kang, "A Framework for Dynamically Converting Components to Web Services," in *Proc. of 3rd ACIS Int. Conf. on Software Engineering Research, Management and Applications*, Michigan, USA, 2005. 431–437.
[45] Z.J. Wang, X.F. Xu, D.C. Zhan. "Normal Forms and Normalized Design Method for Business Service," in *Proc. of IEEE Int. Conf. on e-Business Engineering*, Beijing, China, 2005. 79–86.

Zhongjie Wang was born in China on November 20, 1978. He is now a lecture in computer application technology in School of Computer Science and Technology at Harbin Institute of Technology (HIT), China. He received B.S. Degree, M.S. Degree and Ph.D. Degree in the Department of Computer Science and Engineering in Harbin Institute of Technology in 2000, 2002 and 2005 respectively. His research interests include software engineering, software reuse, software reconfiguration, software component related techniques.

Xiaofei Xu was born in China on November 2, 1962. In 1978, he started his study in Harbin Institute of Technology. He received B.S. Degree, M.S. Degree and Ph.D. Degree in the Department of Computer Science and Engineering in Harbin Institute of Technology in 1982, 1985 and 1988 respectively. And he was awarded as one of the Outstanding Chinese Doctor by the Education Minister of China in 1990.

Prof. Dr. Xu is now a professor and dean of School of Computer Science and Technology, dean of National Pilot School of Software in Harbin Institute of Technology in China. He is commissioner of China Association of Science and Technology, the standing member of the council of China Computer Federation, member of Expert Group for Discipline of Computer Science and Technology in the Academic Degree Committee of the State Council of China. He is member of the Expert Committee of Chinese National 863 High-Tech R&D Program on CIMS (Computer Integrated Manufacturing Systems), member of the Expert Group for Manufacturing Informational Application Project in the National Key Technology R&D Program. He is vice director of National Standard Technical Committee on Industrial Automation System and Integration, vice chairman of the Council of the China ERP Development and Technology Association. He is also senior member of Society of Manufacturing Engineer (SME) in USA, and was member of German Society of Operation Research. He is also guest doctoral supervisor of Dublin Institute of Technology in Ireland. He has positions in the editorial committees of nine academic journals.

His research fields include intelligent enterprise computing, computer integrated manufacturing systems, databases, management and decision information systems, ERP and supply chain management, e-business, knowledge engineering, etc. In recent years, he has been in charge of more than twenty Chinese national research projects and international cooperation projects, and gotten many research achievements and awards. He has published more than 200 papers in journals and conferences in which more than 70 papers are involved in SCI, EI and ISTP, and three academic books. He has held conference chairman and session chairs in the international conferences for several times.

As a doctoral supervisor, he has supervised more than 30 doctoral students in which 10 students get Ph.D. degree, and more than 30 master students in which 28 students get master degree.

Dechen Zhan was born in China on October 18, 1965. He is a professor in School of Computer Science and Technology at Harbin Institute of Technology (HIT), China. His research interests include computer integrated manufacturing system (CIMS), enterprise resource planning (ERP), decision support systems (DSS), software reuse and reconfiguration, etc.