Research Article

Design Intend Solving: Dynamic Composition Method for Innovative Design Based on Virtual Cloud Manufacturing Resource Generators

Yi-Cong Gao, Yi-Xiong Feng, Jin Cheng, Jianrong Tan, and Zheng Hao

State Key Lab of Fluid Power Transmission and Control, Zhejiang University, Hangzhou 310027, China

Correspondence should be addressed to Feng Yixiong; fyxtv@zju.edu.cn

Received 2 February 2013; Accepted 4 July 2013

Academic Editor: Zude Zhou

Copyright © 2013 Gao Yicongetal. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Recently, there has been growing interest in composition of cloud manufacturing resources (CMRs). Composition of CMRs is a feasible innovation to fulfill the user request while single cloud manufacturing resource cannot satisfy the functionality required by the user. In this paper, we propose a new case-based approach for the composition of CMRs. The basic idea of the present approach is to provide a computational framework for the composition of CMRs by imitating the common design method of reviewing past designs to obtain solution concepts for a new composite cloud manufacturing resource (CCMR). A notion of virtual cloud manufacturing resource generators (VCMRGs) is introduced to conceptualize and represent underlying CCMRs contained in existing CCMRs. VCMRGs are derived from previous CCMRs and serve as new conceptual building blocks for the composition of CMRs. Feasible composite CMRs are generated by combining VCMRGs using some adaptation rules. The reuse of prior CCMRs is accomplished via VCMRGs within the framework of case-based reasoning. We demonstrate that the proposed approach yields lower execution time for fulfilling user request and shows good scalability.

1. Introduction

Cloud manufacturing is nowadays emerging as a major technology for larger-scale collaboration manufacturing. It is a new service-oriented, high efficiency and low consumption, knowledge-based, and intelligent networked agile manufacturing model and technology [1, 2]. It enriches and expands the range of resource sharing and service model in cloud computing, promoted agile, service, green, and intelligent-oriented manufacturing development. Service system and technical architecture of cloud manufacturing are constructed by integrating cloud computing, Internet of things, high performance calculation (HPC), service computing, artificial intelligence (AI), and information-oriented manufacturing.

The increasing demand of users for high quality and timely information is putting product manufacturing under the pressure of adjusting the cost, reliability of products, and effectiveness of manufacturing. A strategy that composes multiple cloud manufacturing resources (CMRs) is essential and provides more benefits to users [3]. CMR composition focuses on synthesizing individual CMRs to create a new resource. It aims to provide more value added resource than a single CMR. A composite cloud manufacturing resource (CCMR) is a partially ordered collection of component CMRs. By choosing appropriate CMRs offered by different CMR providers, specifying their coordination plan, and implementing the plan through an orchestration engine, the CCMR can provide more valuable and complete resource than a single CMR. It can also reuse individual CMR more efficiently.

This paper presents a systematic approach to compose CMRs for the users of cloud manufacturing platform. The present approach employs case-based reasoning (CBR) to provide an overall framework for the composition of CMRs. The cases of existing CCMRs are represented in terms of primitive cloud manufacturing resources (PCMRs) and their relevant functions. The underlying CCMRs of them are then conceptualized and identified as virtual cloud manufacturing resource generators (VCMRGs), which consist of respective
functions and physical building block(s) achieving their functions. These VCMRGs are then stored in a case base for further composition. When a user's request is given, relevant VCMRGs are retrieved from the case base and combined by some adaptation rules to fit the given requirements, producing CCMRs of a desired request. With the notion of VCMRGs, the present approach can effectively generate various new CCMRs, sometimes creative ones, by transferring and/or combining prior CCMRs derived from other CCMRs.

The remainder of this paper is organized as follows. In Section 2, a brief review of existing literature on composition of CMRs in network manufacturing is given. Section 3 describes the representation of CMRs and the case memory organization and indexing scheme is given in Section 4. Section 5 gives a detailed explanation of the proposed composition method. In Section 6, the performance of the proposed method is evaluated and analyzed. Finally, the paper ends with a summary given in Section 7.

2. Review of the Literature

Composition of CMRs primarily addresses the situation of a user's request that cannot be satisfied by any available service, whereas a composite service obtained by combining available services might be used. This has triggered considerable number of research efforts and made CMRs composition problem as an active research area. The concept of cloud manufacturing is proposed recently; most of the previous approaches have researched on composition of cloud computing resources and composition of web services.

An important issue related to web services composition is concerned with computing automatic composition of Web services. Zeng et al. [4] proposed an optimizing service selection at a composite service based on a generic Quality of Service (QoS) model using established linear programming techniques. Later, Zeng et al. [5] presented a QoS-aware middleware platform, called AgFlow, supporting quality driven web services composition. The AgFlow system consisted of a service quality model and two alternative service selection approaches. Agarwal et al. [6] classified Web services as their interfaces and built a composite service through logical and physical composition. In the logical composition step, they supported building a composite service, including conditional branches, by applying contingency planning. In Gekas and Fasli [7], all possible compositions among web services were searched using multiply adjacency matrices N times based on depth first search. Berardi et al., [8] developed Colombo, which was a framework for automatic web services composition. Colombo was not developed for web services composition search but for real time transmission of output values when web services were actually performed through workflow by using messaging technique. Kona et al. [9] used constraint logic programming to discover a Web service and built a composite service. Their method checked whether requested outputs and effects were reachable from the provided inputs and preconditions using available services. Then they built an appropriate composite service based on the reachability. Akkiraju et al. [10] used semantic matching, which considered domain dependant information as well as domain independent information and improved semantic precision of generated composite services by considering semantic ambiguity during the planning process. Li et al. [11] proposed a model that enables a designer to clearly understand the global matching relationship between the key components.

Another issue of web services composition is the models and algorithms for selecting web services composition available. Vaculin and Sycara [12] presented a matchmaking algorithm for retrieval of the top-k collision-free services combinations. The time complexity of the algorithm was \( O(m \log m \cdot n) \), where \( n \) was the number of output of a user request and \( m \) was the maximum number of web services able to produce some output effect in the request. The composition results avoided collision such as unwanted side effects, effect duplications, and contradictory effects. However, they returned only top-\( k \) results. Zhao and Doshi [13] introduced the challenges of the RESTful web services composition and proposed a formal model for RESTful web services composition. McIlraith and Son [14] proposed to use Golog, a high-level logic programming language based on the situation calculus, for creating composite web service. They mapped an OWL-S service model to situation calculus. Golog was responsible for finding a goal-oriented composition plan and executing it as a program on an open agent architecture which enabled the program to communicate with outsourced services on the Web. Bultan et al. [15] modeled constituent web services as Mealy machines, the finite state machines with message queues. The individual web services communicated through asynchronous messages and each of them had a queue for incoming messages. By analyzing the sequence of the conversation messages, they provided an approach for modeling well-formed service composition. Traverso and Pistore [16] treated the planning problem based on model checking. They developed ASTRO tool set that was a platform for web service design and execution with automated composition and monitoring. The tool set took as input the OWL-S service models of available outsourced web services and a composition goal and converted them into state transition systems. The model based planner synthesized the state transition systems to satisfy the goal and automatically generated the executable BPEL process of the new service. Sirin et al. [17] treated web service composition as a goal-oriented hierarchical task network (HTN) planning problem and developed SHOP2 HTN system, which took as input an OWL-S service model that reflected the client requirement and executed the resulting plan over the Web. Berardi et al. [18] expressed a client's service requirement as a finite state machine consisting of activities which can be implemented by external web services. Associated with a state was a set of possible activities the client chooses. The resulting state was determined by the client's action. The state followed by activity was iterated until the final state (e.g., terminal state) is reached. They proposed an automatic composition tool called E-Service Composer that implemented the state machine via BPEL.
3. Representation of Cloud Manufacturing Resources

3.1. Primitive Cloud Manufacturing Resources. PCMRs are provided as basic building blocks for representing and understanding existing CCMRs. In general, there are two kinds of PCMRs: manufacturing resource and manufacturing capability [19]. Manufacturing resource consists of physical manufacturing devices (e.g., CNC machine tools), computing devices, and materials, which are involved during the whole lifetime of production. The other is soft PCMR, including software, manufacturing information, and manufacturing knowledge. Manufacturing capability consists of design capability, simulation capability, experiment capability, management capability, and maintenance capability. Manufacturing capability is a dynamic resource form and bases on the manufacturing resource.

For the needs of our research on CMR composition, we developed resource library as a means for modeling and defining a PCMR. The resource library has all the relevant information about PCMRs such as business parameters of PCMRs, states, execution time, per cost, and function (input/output data). The states of a PCMR are wrapped into four perspectives. Table 1 shows an illustrative example of a PCMR in the resource library of cloud manufacturing platform.

(1) The Availability Perspective. We assume that there is a cloud manufacturing platform of \( N \) PCMRs. Let \( sw_i \) denote the availability of PCMR \( R_i \) at the moment of composition:

\[
sw_i = \begin{cases} 
1, & \text{PCMR } R_i \text{ is available}, \\
0, & \text{otherwise}.
\end{cases}
\]

In this paper, we assume that if \( R_i \) is composited then there is enough execution time for it to finish the task.

(2) The Property Perspective. The property perspective illustrates six properties of resource are defined, which have been emphasized on previous work as the quality evaluation criteria of Web service. Let \( te_i \) denote execution time of PCMR \( R_i \) and \( cp_i \) denote per cost of working time of PCMR \( R_i \). It is clear that

\[
C_i = te_i \cdot cp_i.
\]

Per cost \( cp_i \) is usually fixed but may be changed according to the PCMR provider’s business policy. The execution cost is registered by the PCMR provider. Execution time \( te_i \) is the average time expected for executing a cloud manufacturing task. Individual execution time upon the requests of users varies because the server loads changes. Therefore, the average execution time should be updated continuously by the resource provider.

The reputation of each PCMR \( R_i \) is \( rep_i \) that is the average reputation score of resource evaluated by users. The individual reputation scores are likely to be subjective, but the average score becomes trustable as the total number of the usages increases. The reputation score is measured by

\[
rep_i = \frac{\sum_{j=1}^{m} rep_{ij}}{m},
\]

where \( rep_{ij} \) is the individual reputation scores of \( j \)th usage and \( m \) is the total number of usages.

(3) The Function Perspective. The function perspective identifies the data that are exchanged between the PCMRs. Let \( id_i \) denote the input of PCMR \( R_i \) and \( od_i \) denote the output of PCMR \( R_i \):

\[
id_i = [(ip_1, iv_1), (ip_2, iv_2), \ldots, (ip_l, iv_l)],
\]

\[
o_d_i = [(op_1, ov_1), (op_2, ov_2), \ldots, (op_k, ov_k)],
\]

where \( ip_i \) is the name of \( i \)th input, \( iv_i \) is the value of \( i \)th input, \( op_k \) is the name of \( k \)th output, and \( ov_k \) is the value of \( k \)th output.

(4) The Business Perspective. The business parameter \( b_i \) identifies the organizations (i.e., providers) that offer the resource

\[
b_i = [(bp_1, bv_1), (bp_2, bv_2), \ldots, (bp_h, bv_h)],
\]

where \( bp_h \) is the name of \( h \)th business parameter and \( bv_h \) is the value of \( h \)th business parameter.

3.2. Composite Cloud Manufacturing Resources. CCMRs can be conceptualized and understood by using PCMRs as constructive building blocks. For example, consider the flow diagram of a cam outsourcing shown in Figure 1(a). The user submits an input request of cam outsourcing (RO) into two output actions of material purchase (MP) and structure analysis of the cam (SA) and transmits the two output actions, respectively, to the output action cam manufacturing. This can be regarded as composed conceptually of four PCMRs.

---

**Table 1: Example of a design outsourcing PCMR.**

<table>
<thead>
<tr>
<th>Type</th>
<th>Design outsourcing</th>
</tr>
</thead>
</table>
| Availability    | \( sw_i = \begin{cases} 1, & \text{PCMR } R_i \text{ is available}, \\
|                 | 0, & \text{otherwise}. \end{cases} \) |
| Property        | \( C_i = 300 \text{ RMB/h}, \text{rep} = 0.89. \) |
| Function        | \( id_i = \begin{bmatrix} (ip_1, \text{CAD drawings}) \\
|                 | (ip_2, \text{Stress analysis}) \\
|                 | (ip_3, \text{Process analysis}) \end{bmatrix}. \) |
| Business        | \( b_i = \begin{bmatrix} (bp_1, \text{Company 1}) \\
|                 | (bp_2, \text{Company 2}) \end{bmatrix}. \) |
Cloud manufacturing platform transforms the input data into a material purchase resource. On the other hand, the platform transforms the input data into a structure analysis resource, which is then transformed into process analysis resource (PA) by CAD drawings (CAD-D) and then finally into a manufacture resource (M) by a CNC machine tool (M-T). In this way, by understanding a CCMR in terms of its constituent PCMRs and their relevant functions, we could easily conceptualize how the overall function of the CCMR has been accomplished.

As illustrated in Figure 1(b), a CCMR can be represented as a directed acyclic graph, in which each arc is labeled with an associated PCMR and each node with appropriate resource type. The root and leaf nodes represent, respectively, the input and output(s) and the interior nodes denote the connection (common resource type) between two concatenated PCMRs. The direction of an arc indicates the flow of motion from input to output. With the arc label and two types of nodes, we can identify the associated PCMR and its relevant function (one of its multiple functions).

Existing CCMRs are represented in directed acyclic graphs as mentioned above and stored in the case base of the cloud manufacturing platform. They serve as fundamental cloud manufacturing cases, from which VCMRGs will be extracted. Table 2 illustrates the case representation of a cam outsourcing shown in Figure 1.
3.3. Virtual Cloud Manufacturing Resource Generators. The notion of VCMRGs is introduced to conceptualize and represent underlying composition concepts of existing CCMRs. VCMRGs are virtual entities that are derived from portions of or the whole of CCMRs. They consist of one or more CCMRs and perform a specific function of user’s request from input to output(s). The notion of VCMRGs allows combining conventional building blocks of individual PCMRs with new higher-level conceptual building blocks consisting of several PCMRs. In the present approach, VCMRGs are the actual constructive building blocks that are used to compose desired CMRs.

VCMRGs can be obtained by extracting all the subgraphs from the graph representations of existing CCMRs. For example, from the graph structure of the cam outsourcing CMR in Figure 1(b), we can extract a total of ten sub-graphs, as depicted in Table 1. Three types of VCMRGs are defined according to the graph structure of extracted sub-graphs, as indicated in Figure 2. An S-type VCMRG is derived from a part of existing CCMRs consisting of a single PCMR. A chain of concatenated PCMRs constitutes a C-type VCMRG. A graph structure with multiple outputs forms a G-type VCMRG. Table 3 illustrates each type of VCMRGs that are created and stored in the cloud manufacturing platform.

S-type and C-type VCMRGs have a single input and a single output. G-type VCMRGs involve a single input and multiple outputs. The physical realization of a VCMRG is captured by its graph structure, through which its constituent PCMR(s) and relevant function(s) can be identified. The reference CMRs are a list of existing CCMRs, from which respective VCMRGs are derived. Since a VCMRG can be used in several different CCMRs, a number of existing CCMRs can be listed as reference CMRs. The derivation type denotes that a VCMRG is derived either in part (partial) or in whole (complete) from its reference CMR.

4. Case Memory Organization and Indexing Scheme

The obtained VCMRGs, together with PCMRs and existing CCMRs, need to be stored into a case memory for efficient retrieval. Figure 3 illustrates an example of the case memory provided in the present framework. The case memory is organized in a multilayered structure with different levels of abstraction. The first functional index layer serves as a system index, which consists of pointers for locating the VCMRGs stored in the next virtual case layer. VCMRGs are indexed by

---

Table 3: Examples of virtual cloud manufacturing resource generators (VCMRGs).

<table>
<thead>
<tr>
<th>Type</th>
<th>S-type</th>
<th>C-type</th>
<th>G-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>S0</td>
<td>C0</td>
<td>G2</td>
</tr>
<tr>
<td>Overall function</td>
<td>RO → MP</td>
<td>RO → PA</td>
<td>RO → M</td>
</tr>
<tr>
<td>Number of PCMRs</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

---

Figure 2: VCMRGs of a cam outsourcing CCMR.

---

Figure 3: An example of the case memory provided in the present framework.
their functions (i.e., data types of input and output(s)). Hence, all the VCMRGs with the same function are clustered under the same index and thus can be retrieved in a group at the same time. S-type and C-type VCMRGs, which have a single input and a single output (SISO), are indexed in the primary index. On the other hand, G-type VCMRGs are indexed into the auxiliary index, classified according to the number of outputs: SI2O (single input/two outputs), SI3O, and so forth. In this layer, CCMRs are dealt with from the functional point of view without considering physical realizations. The second is the virtual case layer, wherein obtained VCMRGs are stored as virtual cases. As shown in Figure 3, VCMRGs are stored separately according to their types. Since VCMRGs are defined with their own physical realizations (constituent PCMRs) to accomplish their overall functions, it finds that this layer represents an abstraction level of CCMRs in terms of functions and physical artifacts (e.g., function-structure mapping). The third physical layer is comprised of the device library of PCMRs and the case base of CCMRs. This layer stores physical building blocks and artifacts.

Let us explain the procedure of storing and indexing VCMRGs with the example shown in Figure 2 and the case memory illustrated in Figure 3. The four S-type VCMRGs derived from the CCMR in Figures 2(a)–2(d) are first stored, respectively, as S0–S3 in the S-type virtual case base. Their constituent PCMRs and reference CMRs are also given in the second and third columns. The details of them can be identified by consulting the device library and the case base, respectively. Next, they are indexed in the primary index (SISO) according to their functions (e.g., S3 is under the index PA → M). Similarly, the three C-type VCMRGs from the CCMR in Figures 2(e)–2(g) are stored as C0–C2 in the C-type virtual case base and indexed, respectively, by their overall functions in the primary index (SISO). The three G-type VCMRGs obtained from the CCMR in Figures 2(h)–2(j) are stored in the G-type virtual case base and indexed appropriately in the SI2O auxiliary index.

In the case memory of Figure 3 are also presented some VCMRGs obtained from other CCMRs. VCMRG S4 is derived from a product design resource (Software 1: RO → SA), and VCMRGs S5, C3, C4, and C5 are from PM3. Notice here that VCMRG S4 involves the same PCMR (PM0: structure analysis resource) as VCMRG S1, but they are treated as distinct VCMRGs because their relevant functions are different (S1: SA → PA, S4: RO → SA). It should also be noted that when adding a newly derived VCMRG to the case memory, we have to search the case memory first to see if there already exists the same VCMRG. If the same one is found, only the information on reference CMRs and derivation types of the existing VCMRG will be updated appropriately. For example, consider the VCMRG (tool 1: PA → M) derived from the above cam outsourcing resource. Since the same VCMRG S3 already exists in the S-type virtual case base, only the new reference CMR (and derivation type not shown in the figure) is added to the existing one (see CA2 of S3 in Figure 3).

5. Composition of Cloud Manufacturing Resources: Retrieval and Adaptation

5.1. Composition of Cloud Manufacturing Resources with a Single Output. The composition of CMRs with a single input and a single output begins with the design specification that consists of the desired motion transformation from input to output. Given a specified function of input data transformation \( M_i \rightarrow M_o \), partial cases (VCMRGs) matched with the specified input or output motion are retrieved from the case memory. A set \( P_i \) of input-matched cases is retrieved from the case memory using the functional index \( M_i \rightarrow * \), where * denotes an arbitrary output. Similarly, output-matched cases are retrieved as a set of \( P_o \) using the index \(* \rightarrow M_o \). Among the retrieved sets \( P_i \) and \( P_o \), exactly matched cases, which satisfy both the input and output data, can be obtained by \( P_i \cap P_o \) since exact matches exist in both \( P_i \) and \( P_o \). Such VCMRGs will fulfill the specified function without any adaptation and thus can be potential solution concepts for the given user’s request.

After removing exact matches from the sets \( P_i \) and \( P_o \), we now have only partial matches in both \( P_i \) and \( P_o \). From these partial cases, new design solutions can be generated by combining two VCMRGs, \( P_i \) and \( P_o \) \( (p_i \in P_i \text{ and } p_o \in P_o) \), if such a pair satisfies a compatibility condition that the output of \( p_i \) should be the same as the input of \( p_o \). We
apply the following heuristic adaptation rules to compose new CMRs, preventing duplication or redundancy (overlap) of existing CCMRs.

1. If two compatible VCMRGs have no common reference CMRs, they can be combined at the common node (the output node $P_i$ and input node $P_o$, both of which have a common type) to compose a new CMR, which is illustrated in Figure 4(a). This composition of two VCMRGs that is derived from different existing CCMRs can generate a new VCMRG.

2. If two compatible VCMRGs have any common reference CMR, but they are not connected in the reference CMR, they can be combined to produce a new CMR. Such a combination will yield a new CMR concept that is not originally part of the reference CMR as shown in Figure 4(b).

3. If two compatible VCMRGs have a common reference CMR and they are overlapped in the reference CMR, they should not be combined. This is because a new CMR generated by concatenating such two VCMRGs comprises redundant PCMRs, as illustrated in Figure 4(c).

4. If two compatible VCMRGs have a common reference CMR and they are adjacent (connected) to each other in the reference CMR, they should not be combined. This is because a VFG that is equivalent to a new CMR that will be produced by combining such two VCMRGs already exists in the case memory as an exactly matched case. An example is shown in Figure 4(d), where the new CMR obtained by VCMRG1 + VCMRG2 is identical to VCMRG3 that was also derived from the same reference CMR.

The SISO CMR composition method explained above provides new solution concepts by either searching for exact matches or combining partial matches. It is very useful to combination or transference of prior concepts. Therefore, previous CCMRs are reused in a new user’s request via the notion of VCMRGs.

5.2. Composition of Cloud Manufacturing Resources with Multiple Outputs. The proposed SISO composite process can be utilized to solve the composition problem of CMRs with a single input and multiple outputs (SIMO). New CMRs with multiple outputs can be generated by repeatedly applying the SISO procedure as described in the following steps.

1. Composing stem chains: given an input and multiple outputs, an arbitrary output is first selected. Then the SISO composite procedure is executed to compose chains, called stem chains, with the specified input and the chosen output. A number of chains (design alternatives) will be generated in this step, each of which may serve as a stem chain for further composition.

2. Composing and merging branch chains: another output is selected and then the SISO composite procedure is invoked again to construct a branch chain connecting the selected output to the stem chain. The branch chain can be merged to the stem chain
at a branching node (the original input node and the interior nodes of the stem chain can be possible branching nodes). The SISO composite procedure is carried out for every pair between the chosen output and branching nodes serving as an input, which will give rise to a variety of topologies of new CMRs.

(3) Composing and merging other branch chains: step 2 is repeated for the other outputs until all the outputs are connected to the new CMR being composed. As a new branch chain is merged to the new CMR being constructed, its interior nodes also become possible branching nodes at the next stage.

In the procedure explained above, the order of output selection causes different topologies of generated CMR. Hence, the above procedure should be performed for each permutation of the outputs to produce all possible topologies. In this process, some duplicated CMRs may be generated and thus graph isomorphism test is necessary to detect and discard duplicate ones.

When merely exact matches need to be retrieved directly, the auxiliary index (e.g., SI2O, SI3O, etc.) can be utilized without invoking the SIMO composite procedure. Notice, however, that such exact matches are also suggested as solution concepts by the above SIMO composite procedure. Each chain of a G-type VCMRG is also stored as an S-type or a C-type VCMRG. The composition of such component chains will produce the original G-type VCMRG. For example, the CCMR of S0 in Figure 2(a) and C0 in Figure 2(e) will yield G1 in Figure 2(i).

6. Experimental Results and Analysis

Generally, a precise evaluation for a CMR composition method includes analysis of CCMRs built by the method and measurement of the execution time of the whole composition process. Based on this observation, we have conducted three experiments, as shown in Table 4. The proposed method is compared to the experimental results of the other two methods in [20, 21].

We have implemented our method and the reference methods using Java and then measured the elapsed time to find the composition using the time stamp of the system. We randomly generated experimental data which conforms to the conditions of Table 4.

In Experiment 1, we measure the time elapsed to find every CCMR by increasing the number of PCMRs from 50 to 250. Similarly, we have conducted Experiment 2 and Experiment 3 after changing the size of VCMRG and the number of user’s requests, respectively. Tables 5–7 show the experimental results.

Table 5 shows that the more PCMRs there are, the more time it takes to find CCMRs. The size of PCMRs grows larger as the number of available CMRs increases in both methods. Although the size of PCMRs grows larger and the number of CCMRs discovered increases, the elapsed time to find available CCMRs does not increase rapidly.

Table 6 shows that the elapsed time to find available CCMRs decreases as the size of the VCMRG grows larger. This result is caused by fixing the number of PCMRs to 150. In the reference methods, the size of the PCMR is decided based on the number of available CCMRs and the user’s requests of those CCMRs. VCMRGs are not included in the reference methods. Although the size of VCMRGs increases, the size of the PCMRs is unchanged.

Table 7 shows that the elapsed time to find available CCMRs increases as the number of input and output parameters increases. The reference methods search every composition between the input and output concepts of a user’s request. Therefore, as the number of parameters of a user’s request increases, the number of CCMRs to be discovered increases, resulting in the increment of elapsed time. Similarly, the number of CCMRs to be discovered also increases in the proposed method as the number of parameters increases. However, elapsed time increases only slightly in our method.
Table 7: Experiment 3: elapsed time to the number of user’s requests.

<table>
<thead>
<tr>
<th>Number of user’s requests</th>
<th>Elapsed time, $t$ (ms)</th>
<th>The proposed method</th>
<th>The method of [20]</th>
<th>The method of [21]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>18</td>
<td>62</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>220</td>
<td>188</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>301</td>
<td>262</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>23</td>
<td>349</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>25</td>
<td>609</td>
<td>427</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>1142</td>
<td>761</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>23</td>
<td>1528</td>
<td>1384</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>24</td>
<td>1671</td>
<td>1151</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>23</td>
<td>1843</td>
<td>1605</td>
<td></td>
</tr>
</tbody>
</table>

since the size of VCMRG where the searching is performed is relatively tiny.

7. Conclusion

This paper presented a case-based approach for the composition of CMRs. VCMRGs are proposed to identify and conceptualize underlying CCMRs of existing CCMRs. Such identified VCMRGs are stored in a case memory together with existing CCMRs and indexed by their functions for efficient retrieval. CCMRs that satisfy user’s requests are composited by combining VCMRGs that are retrieved from the case memory. VCMRG provides new types of conceptual building blocks that may be composed of several PCMRs as well as individual PCMRs, through which subassemblies of previous designs can be easily delivered as a whole (in a group) to new CCMRs. VCMRGs serve as linkages between functions and structures, and thus they facilitate function-to-structure mapping in CMRs composition. In addition, they can help represent common functionalities shared by several CCMRs and capture different usage of any PCMR resulting from its multiple functions. The present approach could provide a computational tool for the common practice of CCMRs. Although the present approach can effectively realize the generation method of combination of prior CCMRs in CMRs composition, the efficiency of the approach depends on the number of PCMRs and VCMRGs. Therefore, the cloud manufacturing platform needs to continuously input new PCMRs. The limitations of the proposed method will be fixed by future research.

Conflict of Interests

The authors declare that they have no conflict of interests.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant no. 51205347 and no. 51275459), the National Natural Science Fund for Excellent Young Scholars of China (no. 51322506) and the China Postdoctoral Science special Foundation (Grant no. 2013T60585).

References


