Mass Customization Oriented and Cost-Effective Service Network

Zhongjie Wang, Xiaofei Xu, Xianzhi Wang

Mass Customization Oriented and Cost-Effective Service Network

Zhongjie Wang, Xiaofei Xu and Xianzhi Wang

School of Computer Science and Technology, Harbin Institute of Technology
No.92 West Dazhi Street, Harbin, China 150001
{rainy, xiaofei, xianzhi.wang}@hit.edu.cn

Abstract. Traditional service composition approaches face the significant challenge of how to deal with massive individualized requirements. Such challenges include how to reach a tradeoff between one generalized solution and multiple customized ones and how to balance the costs and benefits of a composition solution(s). Service network is a feasible method to cope with these challenges by interconnecting distributed services to form a dynamic network that operates as a persistent infrastructure, and satisfies the massive individualized requirements of many customers. When a requirement arrives, the service network is dynamically customized and transformed into a specific composite solution. In such way, mass requirements are fulfilled cost-effectively. The conceptual architecture and the mechanisms of facilitating mass customization are presented in this paper, and a competency assessment framework is proposed to evaluate its mass customization and cost-effectiveness capacities.

Keywords: service network; service composition; mass customization; cost-effectiveness; competency assessment

1 Introduction

The emergence of service-oriented technologies and trends, e.g., cloud computing, SoLoMo (Social, Local and Mobile), virtualization, and Internet of Things, have promoted an increasing number of software services on the Internet. In addition, there are now various offline physical and human services that are virtualized and connected to the Internet and collaborate with online software services. Such a proliferation of available services has led to a situation where it is time-consuming and costly to select the appropriate service from an extensive range of candidates when building a coarse-grained composite solution to satisfy individualized customer requirements.

Within the service computing domain, this issue is both a traditional and popular research topic termed *service composition*. Although there has been much research in recent years, it has been insufficient in the face of *mass customization*. To lower service composition and delivery costs, it is better to build a standard solution and provide it to all customers. However, due to the divergence of requirements of different customers, such standardization-based strategies consequentially lead to lower de-

grees of customer satisfaction. In contrast, if multiple fully personalized solutions are constructed based on each customer's preferences, then the cost is bound to significantly increase. Therefore, it is critical to look for a tradeoff between a fully generalized solution and multiple individualized ones, and to balance the costs and benefits of a composition solution(s). In addition, the solutions generated by current approaches are usually temporary, i.e., after the corresponding requirements have been fulfilled, they are released and no longer exist. This action further increases costs.

Let us take a referential idea from the Internet: consider the scenario that users make end-to-end communications via the Internet. It is not necessary to establish a direct connection between their computers but a virtual link is dynamically established by a routing mechanism based on the infrastructure. After the communication finishes, this link is disconnected. The persistent Internet infrastructure could satisfy any type of individualized communication demands and end users do not necessarily know the details of the complex protocols.

Using this basic philosophy as a reference, we propose the concept of a "Service Network" (SN). It could be considered as business-level persistent infrastructure in the form of interconnections between distributed services, and able to satisfy a large number of customers with customized requirements. The nodes in a SN would include various services (e.g., e-services, human services, information, and resources), and the interconnections between them are information exchanges and functional invocations following interoperability protocols such as SOAP and REST. As shown in Fig. 1, services are deployed on different servers and they are logically connected under the support of Internet.

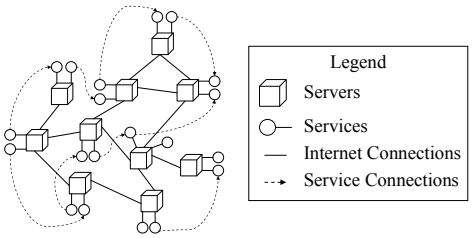

Fig. 1. Service Network: Interconnections between Distributed Services

The SN concept is not a novel idea. Below are two cases from the real world:

(1) *ifttt* (If-This-Then-That, http://ifttt.com). This is a website that aims to connect services from different websites (e.g., APIs of *twitter*, *facebook*, and *instagram*) using an event triggering mechanism to realize cross-domain service invocation. Users set up an "*IF*…*THEN*..." task on ifttt.com, and when the condition after *IF* is satisfied, the service after *THEN* is triggered. In this way distributed services on the Internet are interconnected as a virtual service network.

(2) *Alibaba* service eco-system. *Alibaba* is a full-scale e-Business provider, whose services include alibaba.com (a B2B platform), tmall.com (a B2C platform), taobao.com (a C2C platform), ju.taobao.com (a Groupon-like platform), alipay.com (a 3rd-party payment service), e56.taobao.com (a 3rd/4th-party logistics service), and

etao.com (product search service). They are connected as a large service network to jointly fulfill the individualized requirements of millions of buyers and sellers.

Just as the objective of the Internet is more than just the fulfillment of a single demand by two specific users, the purpose of an SN is not to fulfill just one requirement raised by one specific customer. On the contrary, any customers could utilize it for their own purposes. Of course, requirements can vary greatly, so an SN should adapt itself to the requirements (on QoS and function) of mass customers by dynamic "transformations". The more individualized and diverse the requirements an SN can satisfy, the higher its competency to facilitate mass customization. In economic terms, an SN should be "cost-effective", i.e., the sum of construction, maintenance, and customization costs should be below the total benefit gained from mass customization.

This paper is organized as follows. In section 2 related works are introduced, and the similarities and differences between the philosophies of traditional service composition, SaaS, and SN are clarified. In section 3, the conceptual architecture of an SN is described and formally defined. Section 4 explains how an SN facilitates mass customization, and section 5 shows a competency assessment framework and corresponding metrics. Finally is the conclusion.

2 Related Work

Mass Customization (MC) [1] originated in the production domain. Similarly, in the software engineering domain, methods such as software product line and reuse-based software engineering (RBSE) emphasize the philosophy of utilizing standard and reusable fine-grained software components to rapidly build applications in one domain, essentially realizing the mass production and customization of software products [2]. Later, ideas from MC and RBSE were imported into the services computing domain and became a key analysis and design approach in pursuing the mass customization of service-oriented systems using techniques like loosely-coupled architecture, autonomic agent, dynamic workflow, and service family [3][4].

Service discovery, selection, and composition [5][6] play critical roles in constructing coarse-grained service solutions that meet individualized requirements. Applicable services are selected from candidates, then potential composite solutions are generated and evaluated, and the most appropriate one is delivered to the customers [7]. In addition to IOPE (Input-Output-Preconditions-Effects) and QoS [8], the customer's preferences and context are addressed to look for an exact match between composite solutions and customer requirements [9][10]. At present, research on this issue is mainly based on AI planning techniques, i.e., initial and expected states, and the semantics of candidate services are formally described. A planner with reasoning capacity is then employed to look for a composite path that transforms the initial state into the expected one by back-chaining or forward-chaining policies [11]. Semantic querying and reasoning are the pivotal techniques used in the process.

Another popular approach is to look for the underlying pattern of each customer from his/her historical service usage records using data mining techniques. Personal-

ized solutions are then built following the identified patterns [12][13]. This method is suited to a scenario where customers do not explicitly state their preferences.

Software as a Service (SaaS) is another successful practice in boosting service mass customization [14]. In SaaS, a meta-data model is used to define variability in the data layer, business logic layer, and user interface layer, and each tenant makes personalized configurations on these variability points [15]. In this way, many personalized requirements can be facilitated by one software instance, and the personalized performance of different tenants is ensured by the scalable architecture [16]. However, the services in a SaaS are largely designed, developed, deployed, and runtime provisioned by the SaaS operator itself, and many services distributed on the Internet are seldom used due to reliability considerations.

It would appear that the distinctions of SN are twofold:

(1) It transforms the "centralized service development, maintenance, and evolution" policy adopted in SaaS into "the utilization and aggregation of massive distributed services on the Internet to form a dynamic network structure" policy, thereby extending the scope and flexibility of mass customization; and

(2) It transforms the "one-requirement-oriented temporary solution" policy adopted in traditional service composition approaches into the "massive-requirement-oriented persistent solution", thereby improving the cost-effectiveness of mass customization.

3 Concept and Architecture of an Service Network

An SN is essentially a combination of multiple composite solutions, each of which is established in terms of one customer requirement. Although superficially it appears to be quite complicated and redundant, it has a higher fitness, i.e., when one requirement arrives, it automatically looks for a sub-network and provides it to the corresponding customer. Each solution is the equivalent of a traditional service composition algorithm.

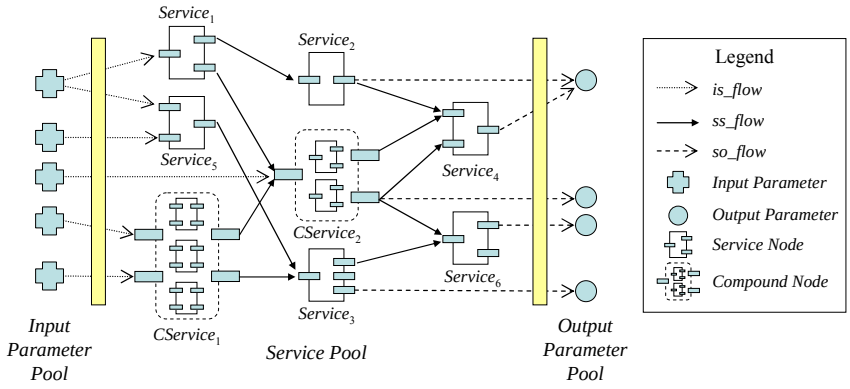

Fig. 2. Conceptual Architecture of a Service Network

As shown in Fig. 2, the structure of an SN looks like an artificial neural network. The left-most part is the input information obtained from three sources: the customers, obtained automatically from the context of customers, or from customers' historical records. The right-most part is the output information expected by customers. In the middle there are multiple layers, with each layer containing multiple services, and the services in the different layers are connected by parameter passing. The input is transformed into the output layer by layer.

An SN is defined by

SN=(IN_Pool, OUT_Pool, $Service\_Pool$, IS_Flow, SS_Flow, SO_Flow),

where

- IN_Pool is the input of an SN and contains a set of parameters. Each parameter $in\_param \in IN\_Pool$ is a term from the domain ontology. For clarity, we suppose that all parameters are atomic and they are mutually independent, i.e., there are no semantics overlaps.
- OUT_Pool is the output of an SN and also contains a set of parameters. Different from in_param, each parameter $out\_param \in OUT\_Pool$ may be either atomic or compound, and there may be semantic overlaps between different parameters.
- $Service\_Pool$ is the main body of an SN and contains of a set of service nodes. For each node, $service$={FD, IN_slots, OUT_slots, QoS_params}, where FD is the semantic description (in the form of ontology), IN_slots is a set of slots, and each slot represents an input parameter of $service$, OUT_slots is a set of slots, and each slot represents an output parameter, and QoS_params is a set of quality parameters. Further, $\forall in\_slot_i \in IN\_slots$ is defined by in_slot_i=(in_param_i, FD_i), where in_param_i is the name of the input parameter and FD_i is the ontology description of semantics. So does $\forall out\_slot_i$=(out_param_i, FD_i)$\in OUT\_slots$.

Note that there is a special type of service node called *compound service* (CS), denoted by CS={$service_1$, $service_2$, …, $service_n$}. For $\forall service_i$, $service_j \in CS$, they have the same FD, IN_slots, and OUT_slots but the value of quality indicators in QoS_params might be different. For example, the nodes $CService_1$ and $CService_2$ are compound services, shown in dotted rounded rectangles.

- IS_Flow={is_flow} is the connections between IN_Pool and $Service\_Pool$. Further, is_flow=$in\_param_i \rightarrow service_j.in\_slot_{jl}$ indicates the transferring of the input parameter $in\_param_i \in IN\_Pool$ to the in_slot_{jl} of a $service_j$.
- SS_Flow={ss_flow} is the connections between service nodes, and ss_flow=$service_i.out\_slot_{ik} \rightarrow service_j.in\_slot_{jl}$ indicates the transferring of the output parameter out_slot_{ik} of service $service_i$ to the input parameter in_slot_{jl} of $service_j$.
- SO_Flow={so_flow} is the connections between $Service\_Pool$ and OUT_Pool, and so_flow=$service_j.out\_slot_{jl} \rightarrow out\_param_i$ indicates the transferring of the output parameter out_slot_{jl} of $service_j$ to the output parameter $out\_param_i \in OUT\_Pool$.

For $\forall in\_slot_{ik} \in service_i$, there must be one or multiple flows pointing to one in_slot of a service node. Such a flow is either an is_flow from IN_pool or an ss_flow from another service node. When the SN is customized, on most occasions only one flow

will be selected and take effect. However, if the customer expects higher reliability, multiple flows may take effect simultaneously, indicating that in_slot_{ik} has multiple data sources (also called *redundancy*). Taking Fig. 2 as an example, the input parameter of $CService_2$ has three distinct sources ($service_1$, *in_pool*, and $CService_1$). Similarly, with $\forall in\_param \in IN\_pool$, there is at least one *is_flow* pointing out from it.

For $\forall out\_slot_{jl} \in service_j$, there may be zero, one, or multiple flows pointing out from one out_slot_{jl} of a service node. Such a flow is either a *so_flow* to *OUT_pool* or an *ss_flow* to another service node. When there are no flows pointing out from the out_slot_{jl}, this indicates that this output parameter is trivial and not used by the SN. Similarly with $\forall out\_param \in OUT\_pool$, there is at least one flow *so_flow* pointing to it. Furthermore, a flow pointing directly out from an $in\_param \in IN\_pool$ to and $out\_param \in OUT\_pool$ is illegal.

4 How Service Network Supports "Mass Customization"

This section describes the SN mechanisms that support "mass customization". Based on the descriptions in section 3, there are two "transformation" mechanisms facilitating mass customization, i.e., (M_1) the dynamic selection of service nodes, and (M_2) the dynamic selection of flows. The variable service nodes and flows are both defined as the "*features*" of an SN, each of which has a limited scope and density of customization. Metaphorically speaking, a feature is like a joint of a human body and its customization scope is the joint's degree of freedom. Table 1 lists a set of customizable features and their customization scope. Focusing on a personalized requirement, the customization of an SN is the process of selecting a specific value for each feature from its customization scope, indicating that a subset of service nodes and a subset of flows are identified, and the SN is transformed into a composite solution.

Table 1. Customizable Features of an SN

Feature	Sub-Feature		Customization Scope
Functionalities	F_1	(M_1) Input parameter in_param_i	{*Selected*, *Not Selected*}
	F_2	(M_1) Service node $service_i$	{*Selected*, *Not Selected*}
	F_3	(M_2) The source of an in_slot_{ik} of $service_i$	$\{is\_flow=*\rightarrow service_i.in\_slot_{ik}\} \cup \{ss\_flow=*\rightarrow service_i.in\_slot_{ik}\}$
	F_4	(M_2) The source of an out_param_j in *OUT_Pool*	$\{so\_flow=*\rightarrow out\_param_j\}$
QoS	Q_1	(M_1) The selected services of a CS_i	$\{service_{i1}, \ldots, service_{in}\}$
	Q_2	(M_1) The number of selected services of a CS_i	$\{1, 2, \ldots, \lvert CS_i \rvert\}$
	Q_3	(M_2) The number of sources of an in_slot_{ik} of $service_i$	$\{1, 2, \ldots, \lvert\{is\_flow=*\rightarrow service_i.in\_slot_{ik}\} \cup \{ss\_flow=*\rightarrow service_i.in\_slot_{ik}\}\rvert\}$
	Q_4	(M_2) The number of sources of an out_param_j in *OUT_Pool*	$\{1, 2, \ldots, \lvert\{so\_flow=*\rightarrow out\_param_j\}\rvert\}$

It is easy to imagine that, in terms of a specific requirement, there might be multiple possible results, each of which could fully satisfy the requirement. Therefore, besides the value assignment for each feature, the customization process should also find the "best" solution from these possibilities. We define it as a combinatorial optimization problem following a "*just-enough*" policy [17]. The following is the mathematical model.

Input:

- An SN;
- Input parameters provided by customer: $\{req\_in\_param\} \subseteq IN\_Pool$;
- Output parameters expected by customer: $\{req\_out\_param\} \subseteq OUT\_Pool$;
- QoS expectations of customer: $\{req\_QoS\_param\}=\{T=TValue, C=CValue, R=RValue\}$, i.e., time, price, and reliability, respectively.

Output: *bp*, a sub-graph of *SN* in which each feature has been assigned a value to.

Decision Variables:

- x_i=0/1 indicates whether $service_i$ is selected;
- If x_i=1 and $CService_i$ is a compound node, then y_{ik}=0/1 indicates whether $service_{ik}$ is selected;
- $z_{ip,jq}$=0/1 indicates whether out_slot_{ip} of $service_i$ is connected to in_slot_{jq} of $service_j$;
- $v_{l,jq}$=0/1 indicates whether in_slot_{jq} of $service_j$ is connected with in_param_l;
- $u_{ip,l}$=0/1 indicates whether out_param_l is connected with out_slot_{ip} of $service_i$;
- $f_T(bp)$, $f_C(bp)$ and $f_R(bp)$ are the calculating functions that compute the global Time, Cost, and Reliability, respectively, of the generated *bp* according to its process structure [18].

Objective Function:

The generated process *bp* satisfies the customer requirement as close as possible, and if no such process can be found, the output is *null*, i.e.,

$$\min y = F(pr, bp) = \left(pr.T - f_T(bp), pr.C - f_C(bp), pr.R - f_R(bp)\right)^T$$

$$s.t.:\ pr.T - f_T(bp) \geq 0,\ pr.C - f_C(bp) \geq 0,\ pr.R - f_R(bp) \leq 0$$

Solving Strategies:

- Phase 1: Pruning

According to $\{req\_in\_param\}$ and $\{req\_out\_param\}$, the initial *SN* is pruned, i.e., (1) $\forall in\_param_i \in IN\_Pool \setminus \{req\_in\_param\}$ and $\forall is\_flow=\{in\_param_i \rightarrow *\}$ are removed from the SN; (2) $\forall out\_param_j \in OUT\_Pool \setminus \{req\_out\_param\}$ and $\forall so\_flow=\{* \rightarrow out\_param_j\}$ are removed from the SN; (3) recursively check each service node from left to right of the SN, examine each in_slot_{ip} of each $service_i$, if there are no flows pointing to in_slot_{ip}, then $service_i$ and all related flows are removed from the SN; (4) check the remaining SN, and if it cannot produce any output parameters, then the requirements cannot be fulfilled by the initial SN and *NULL* is returned.

- Phase 2: Optimization

Based on the pruned *SN*, a multi-objective programming approach is employed to solve the combinatorial optimization problem.

5 Competency Assessment of Service Network

In reality, the competency of an SN is limited; in other words, not all individualized requirements can be satisfied by an SN, and only a certain number of requirements can be simultaneously satisfied. Even if a requirement could be satisfied, there are associated costs that must be paid. This section puts forward a set of indicators to assess the competency of an existing SN, with corresponding metrics.

5.1 Competency Assessment Framework (SN-CAF)

The competency of an SN is assessed from two aspects: capacity of mass customization and cost-effectiveness. Figure 3 shows the complete assessment framework containing eight atomic indicators.

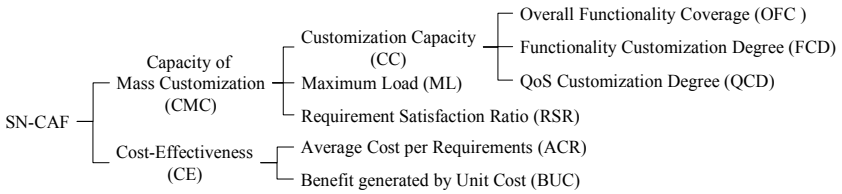

Fig. 3. Competency Assessment Framework of a Service Network

5.2 Assessment of the Capacity of Mass Customization (CMC)

The capacity of mass customization (CMC) is measured by looking at "customization" capacity and "mass" capacity. The former refers to the scope of functionalities and QoS that could be customized in an SN, and the latter refers to the scale or the number of requirements that could be simultaneously fulfilled by one SN. From a statistical point of view, CMC may be indirectly measured by the proportion of satisfied requirements relative to the total arriving requirements.

More specifically, we use five fine-grained indicators, the first three, Overall Functionality Coverage (OFC), Functionality Customization Degree (FCD), and QoS Customization Degree (QCD), measure the "customization" capacity, the fifth (Maximum Load, ML) measures the "mass" capacity, and the final one (Requirement Satisfaction Ratio, RSR) is a statistical measurement.

(1) Overall Functionality Coverage (OFC)

OFC refers to the degree of functionality coverage of an SN relative to the business domain it belongs to. It characterizes the richness of functionality of an SN and is measured by the proportion of the ontology covered by an SN relative to the holistic

ontology of the domain. Because the size of the domain ontology is difficult to estimate, we use the number of complete ontology covered by the SN as the metrics, i.e., $OFC(SN) = \bigcup_{service_i} service_i.FD$.

The greater the functionality of an *SN*, the higher the diversity of its functions, and thereby the higher possibility that it might fulfill a varying number of functions, and the broader range of choices a customer may have to customize his/her functional preferences.

(2) Functionality Customization Degree (FCD)

FCD is the metrics indicating the degree by which functionalities could be customized. It may be measured by the percentage of customization relative to the total functionalities, and the customization scope of each functionality feature, using the four functionality features listed in Table 1, i.e., (F_1) the selection of input parameters and (F_2) service nodes, (F_3) the selection of the source of an input parameter of a service node, and (F_4) the selection of the source of an output parameter. The following are detailed metrics to calculate the degree of customization:

For F_1, we check each input parameter whether it is either optional or mandatory. A mandatory input parameter has to be selected when the SN is used, so it cannot be customized, and the opposite is true when it is optional. The pruning strategy (mentioned in section 4) is used to delete in_param_i and all its related service nodes and output parameters from the SN, and if no output parameters remain then in_param_i is mandatory; otherwise it is optional. Fig. 4 schematically illustrates an example pruning process, in which the deletion of the first input parameter leads to the deletion of $Service_1$, $Service_2$ and $Service_5$, but not incurs to the disappearances of the output parameters, so this input parameter is optional. $FCD_1(SN)=\frac{\left|\{in\_param_i : \text{optional}\}\right|}{\left|IN\_Pool\right|}$ is used to measure the percentage of customizable input parameters in *IN_Pool* of SN.

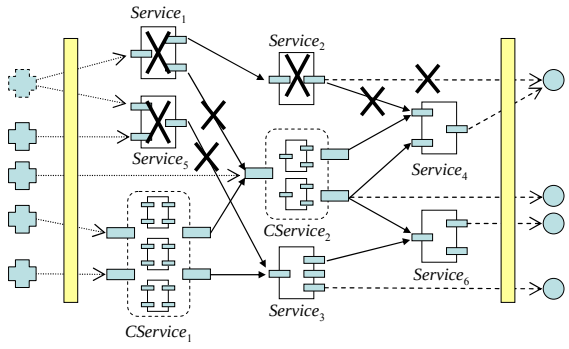

Fig. 4. The Pruning of the SN

For F_2, we check each service node whether it is optional or mandatory. The same pruning strategy is used to prune $service_i$ and all its related service nodes and output parameters to check whether $service_i$ is optional or mandatory. Then $FCD_2(SN)=$

$\frac{\left|\{service_i : \text{optional}\}\right|}{\left|Service\_Pool\right|}$ measures the percentage of customizable service nodes in *Service_Pool* of the SN.

For F_3, we calculate the number of sources of an in_slot_{ik} of $service_i$ and that the number is larger than 1 implies that this parameter can be customized. Then $FCD_3(SN) = \frac{\sum_{service_i} \left|\left\{in\_slot_{ik} : \left|\{flow : flow = * \rightarrow in\_slot_{ik}\}\right| > 1\right\}\right|}{\sum_{service_i} \left|IN\_slots_i\right|}$ measures the percentage of customizable input parameters of services in the SN.

For F_4, we calculate the number of sources of an out_param_j and that the number is larger than 1 implies that this parameter can be customized. Then $FCD_4(SN) = \frac{\left|\left\{out\_param_j : \left|\{flow : flow = * \rightarrow out\_param_j\}\right|\right\} > 1\right|}{\left|OUT\_Pool\right|}$ measures the percentage of customizable output parameters in *OUT_Pool* of the SN.

Synthesizing the four together is represented by $FCD(SN) = \frac{1}{4}(FCD_1(SN) + FCD_2(SN) + FCD_3(SN) + FCD_4(SN))$.

(3) QoS Customization Degree (QCD)

QCD is defined as the overall scope in which the global QoS of the customized solutions can vary. In terms of time, cost, and reliability, the measurements are different.

First we transform the SN into the form of a service process using the following steps:

- Construct a *Start* activity and an *End* activity;
- Construct an *activity* for each simple service node, and for each candidate service in each compound service node;
- For the activities transformed from compound service nodes, add an *or-split* before them and an *or-merge* after them;
- If there is an *ss_flow* pointing from an input parameter in *IN_Pool* to a service *node*, construct an arrow connecting *Start* and the corresponding *activity*; similarly, place arrows between two activities, and between an *activity* and *End*;
- If there are multiple arrows between two activities, keep one only;
- If multiple flows point to the same *in_slot* of a service node, then label the corresponding arrows *or-merge*;
- If a flow is optional, also label the corresponding arrow *or-merge*.

Figure 5 provides an example process based on the SN shown in Fig. 2.

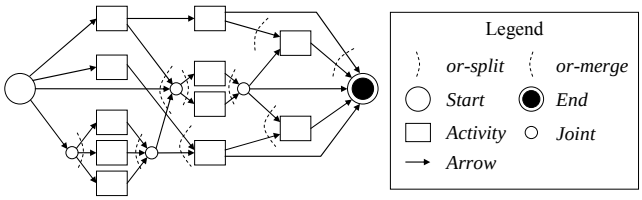

Fig. 5. Service Process based on an SN

Based on the induced process, the following provide accurate measurements of *QCD*(*SN*) for time, cost, and reliability:

[*Time*] Identify the paths with the longest time (*LT*) and shortest time (*ST*) using the Critical Path Method, and then $QCD_T(SN)=[LT, ST]$.

[*Cost*] To measure the upper bound of cost (*UC*), keep a branch with a maximal price for each fragment between *or-split* and *or-merge*, delete all other branches, and then sum the prices of the remaining activities; similarly, keep the branch with the minimal price and obtain the lower bound of the cost (*LC*); then $QCD_C(SN)=[LC, UC]$.

[*Reliability*] To measure the upper bound of reliability (*UR*), keep the process unchanged and calculate global reliability based on the structure of the process, because the original process retains all the redundancy, and the global reliability is maximized. To measure the lower bound (*LR*), for multiple arrows grouped by the same *or-merge*, retain one arrow whose source activity has minimal reliability, and delete all other arrows and related activities; then calculate the corresponding global reliability. Such pruning eliminates all redundancy and retains activities with minimal reliability. Thus, $QCD_R(SN)=[LR, UR]$.

(4) Maximum Load (*ML*)

ML is the maximal concurrency number of requirements that could be simultaneously satisfied by the *SN*. Different requirements may be allocated to different paths, or share the same path.

Suppose the concurrency number of an atomic $service_i$ is $CN(service_i)$. For a compound service node, namely CS_j, its CN increases to $\prod_{k=1}^{|CS_j|} CN\left(service_{jk}\right)$ implying all the atomic services contained in CS_j could be concurrently utilized to satisfy requirements.

Essentially, the value of *ML* lies on the minimal concurrency number of service nodes, which are mandatory and cannot be pruned (this was mentioned in the measurement of FCD). So, $ML(SN)=\min SN(service_i)$ where $service_i \in \{service: mandatory\}$.

(5) Requirement Satisfaction Ratio (RSR)

RSR is a statistics metric and is not directly related to the structure of SN. It is the ratio that the number of satisfied requirements relative to the total arriving requirements during a period of time. This ratio indirectly reflects the competency of mass

customization. If the ratio is low, then there are inevitably deficiencies in the SN and it is necessary to enhance it immediately.

5.3 Assessment of Cost Effectiveness

An SN is cost-effective when the benefit generated by satisfying the individualized requirements of a large number of customers is greater than the sum of the costs incurred during the full lifecycle of the SN, including construction costs, maintenance costs, and customization costs. Otherwise, the SN becomes worthless. Put another way, the more times that an SN is used and the greater the amount and frequency of individualized requirements it could satisfy, therefore the greater significance it has, and the greater benefit it could bring to the operator of the SN.

There are three associated costs:

- Construction Costs (NC): the cost of selecting the appropriate candidate services, negotiating with the providers of these services, and connecting them together to form an initial SN. It is paid for during the initialization of infrastructures and is a "sunk cost" or "fixed investment".
- Maintenance Costs (MC): the cost of provisioning, evaluating, and continuous enhancing (discussed further in section 6) per unit of time (e.g., one month). Such costs are not directly related to the requirements but are seen as similar to daily operation costs.
- Customization Costs (CC): the cost of customizing all of the features of an SN to satisfy a specific requirement.

Suppose there are n possible requirements based on forecasting and historical records, and the "Willing to Pay (WTP)" of the i-th requirement is WTP_i. If they are satisfied by traditional service compositions, i.e., each one is provided with a directly-constructed solution, and the construction cost of the i-th solution is DC_i, then the total benefit is: $DB = \sum_{i=1}^{n} \left(WTP_i - DC_i \right)$. If a service network is constructed to satisfy n requirements and they are distributed in m months, then the total benefit is $SNB = \sum_{i=1}^{n} \left(WTP_i - CC_i \right) - NC - m \times MC$. Further, the cost-effectiveness of the constructed SN is denoted by $CE(SN) = \frac{SNB - DB}{DB}$, indicating the percentage of the increased benefit that the SN produces compared with a traditional approach.

In addition to CE, there are another two meaningful metrics:

(1) Average Cost per Requirement (ACR), i.e.,

$$ACR(SN) = \frac{1}{n}\left(\sum_{i=1}^{n} CC_i + NC + m \times MC \right);$$

(2) Benefit generated by Unit Cost (BUC), i.e.,

$$BUC(SN)=\frac{SNB}{n\times ACR(SN)}=\frac{\sum_{i=1}^{n}WTP_i}{\sum_{i=1}^{n}CC_i+NC+m\times M}-1.$$

Ideally, if an SN is cost-effective, ACR(*SN*) and BUC(*SN*) will vary, with *ACR*(*SN*) being extremely high and *BUC*(*SN*) negative during the stage when an SN is initially constructed; as the number of arriving requirements increases, *ACR*(*SN*) drops gradually and reaches a stable level, and *BUC*(*SN*) ascends gradually and becomes positive. As a comparison, *ACR* and *BUC* in a traditional service composition approach remains stable with small fluctuations.

6 Conclusions

By focusing on the mass customization of services, this paper analyzed the deficiencies of traditional SaaS and service composition approaches, and then proposed a "Service Network" as a means to solve the contradictions between "mass-oriented standardization" and "individual-oriented personalization". An SN is constructed and maintained as a persistent existing infrastructure, to be transformed into various concrete solutions to satisfy considerable individualized customer requirements. This is achieved under the support of a set of customized features such as service nodes, flows between services, and varied QoS in compound service nodes. In addition to mass customization competency, this study also highlighted "cost-effectiveness" as an important factor, an aspect that has been ignored in previous research.

Accompanied by the continuing growth in Internet services, customer requirements are becoming increasingly diversified and their granularity tends to large-grained. In such circumstances, cross-organization and cross-regional service collaboration have been the dominant trends in many service industries. Our study provides some perspective references regarding the research and practices in this respect.

Acknowledgment

The work in this paper is supported by the projects funded by the Natural Science Foundation of China (Nos. 61272187 and 61033005).

References

1. Silveiraa, G.D., Borensteinb, D., Fogliattoc, F.S.: Mass customization: literature review and research directions. International Journal of Production Economics 72(1), 1–13 (2001)
2. Hallsteinsen, S., Hinchey, M., Park, S., Schmid K.: Dynamic software product lines. IEEE Computer. 41(4), 93–95 (2008)
3. Karpowitz, D., Cox, J., Humpherys, J., Warnick, S.: A dynamic workflow framework for mass customization using web service and autonomous agent techniques. Journal of Intelligent Manufacturing. 19(5), 537–552 (2008)

4. Moon, S.K., Shu, J., Simpson, T.W., Kumara, S.: A module-based service model for mass customization: service family design. IIE Transactions, 43(3), 153–163 (2010)
5. Hwang, S.Y., Lim, E.P., Lee, et al.: Dynamic web service selection for reliable web service composition. IEEE Transactions on Services Computing. 1(2), 104–116 (2008)
6. Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Transactions on Software Engineering, 30(5), 311–327 (2004)
7. Ardagna, D., Pernici, B.: Adaptive service composition in flexible processes. IEEE Transactions on Software Engineering. 33(6), 369–384 (2007)
8. Cavallo, B., Penta, M.D., Canfora, G.: An empirical comparison of methods to support QoS–aware service selection. In: Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems, pp. 64–70 (2010)
9. Li, Y., Huai, J.P., Sun, H., Deng, T., Guo, H.: Pass: An approach to personalized automated service composition. In: Proceedings of IEEE International Conference on Services Computing, 283–290 (2008)
10. Lin, N., Kuter, U., Sirin, E.: Web service composition with user preferences. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M.(eds) The Semantic Web: Research and Applications. LNCS, vol. 5021, pp. 629–643 (2008)
11. Peer, J.: Web service composition as AI planning: a survey. University of St. Gallen, Switzerland (2005)
12. Gaaloul, W., Baïna, K., Godart, C.: Log-based mining techniques applied to Web service composition reengineering. Service Oriented Computing and Applications. 2(2–3), pp. 93–110 (2008)
13. Tang, R.: An approach for mining web service composition patterns from execution logs. In: Proceedings of the 12th IEEE International Symposium on Web Systems Evolution, pp. 53–62 (2010)
14. Sun, W., Zhang, X., Guo, C.J., et al.: Software as a Service: configuration and customization perspectives. In: Proceedings of IEEE Congress on Services, pp. 18–25 (2008)
15. Mietzner, R., Metzger, A., Leymann, R., Pohl, K.: Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications. In: Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems, pp. 18–25 (2009)
16. Shim, J., Han, J., Kim, J., et al.: Patterns for configuration requirements of Software-as-a-Service. In: Proceedings of ACM Symposium on Applied Computing, pp. 155–161 (2011)
17. Ni, W., He, L., Liu, L., Wu, C.: Commodity-market based services selection in dynamic web service composition. In: Proceedings of the 2nd IEEE Asia-Pacific Service Computing Conference, pp. 218–223 (2007)
18. Jaeger, M.C., Rojec-Goldmann, G., Muhl, G.: QoS Aggregation for Web Service Composition using Workflow Patterns. In: Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conference, pp.149–159 (2004)