International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:7, 2007

A Quantitative Approach to Strategic Design of Component-Based Business Process Models

Eakong Atiptamvaree and Twittie Senivongse

Abstract**—**A new paradigm for software design and development models software by its business process, translates the model into a process execution language, and has it run by a supporting execution engine. This process-oriented paradigm promotes modeling of software by less technical users or business analysts as well as rapid development. Since business process models may be shared by different organizations and sometimes even by different business domains, it is interesting to apply a technique used in traditional software component technology to design reusable business processes. This paper discusses an approach to apply a technique for software component fabrication to the design of process-oriented software units, called process components. These process components result from decomposing a business process of a particular application domain into subprocesses with an aim that the process components can be reusable in different process-based software models. The approach is quantitative because the quality of process component design is measured from technical features of the process components. The approach is also strategic because the measured quality is determined against business-oriented component management goals. A software tool has been developed to measure how good a process component design is, according to the required managerial goals and comparing to other designs. We also discuss how we benefit from reusable process components.

Keywords**—**Business Process Model, Process Component, Component Management Goals, Measurement

I. Introduction

PROCESS-ORIENTED software development is a new software development paradigm in which the application domain is modeled as a business process model (BPM) [1]. A BPM describes the control flow of the operational process of the business with business rules incorporated. Users or business analysts who are familiar with the nature of the business can easily model their business with BPMs. This is opposed to modeling with software models such as UML [2] which requires expertise of software designers. Since BPMs correspond to process flow, software designers can rapidly develop applications by mapping BPMs to a workflow execution language and have them run with a supporting execution engine. The building blocks of this paradigm are software units that are composed together to work according to the BPM. Current software technology such as Web Services technology [3] realizes this paradigm with software building blocks called Web Services that can be composed by using a process execution language called BPEL [4].

By promoting reuse of software units across the design and development of different application domains, process-oriented paradigm is analogous to building applications with traditional software component technology [5]. We therefore see a potential to apply a traditional software component technique to process-oriented software development.

In this paper, we look from the component supplier's point of view and focus on a software component fabrication technique (i.e., how to develop components). Traditionally, component-based software is modeled by using UML class diagram and the model is decomposed into small units in order to implement them as reusable software components. Analogously in process-oriented paradigm, an application domain is modeled by a BPM which can be decomposed into subprocesses called process components [6]. These process components can be reused in the design of the BPMs of other businesses or application domains, and can also be implemented into reusable software units. This idea corresponds to the concept of process patterns [7].

This paper applies a software component fabrication technique presented in [8] to fabricate process components. The technique in [8] starts by modeling an application domain with a UML class diagram and dividing the classes within the diagram into groups, each group referring to a software component (to be implemented). Such grouping can be seen as one way to design software components for the application domain. Technical features are measured from the design, namely intercomponent coupling, intracomponent cohesion, number of components, component size, and complexity. The resulting measurement values are applied onto a mathematical model, called the Business Strategy-Based Component Design (BusCod) model, to determine how well such a software component design can achieve predefined managerial goals (i.e., cost effectiveness, ease of assembly, customization, reusability, and maintainability). We are particularly interested in this technique because it can give quantitative measurement that reflects quality of the design while also considering business strategies.

In this paper, we will model a particular application domain with a BPM and decompose the BPM into process components so that the technique in [8] can be applied. Software designers can try to design process components for a particular domain in different ways (i.e., with different

E. Atiptamvaree is with the Information Systems Engineering Laboratory, Department of Computer Engineering, Chulalongkorn University, Bangkok 10330 Thailand (phone: +66 2 2186991; fax: +66 2 2186955; e-mail: Eakong.A@student.chula.ac.th).

T. Senivongse is with the Information Systems Engineering Laboratory, Department of Computer Engineering, Chulalongkorn University, Bangkok 10330 Thailand (phone: +66 2 2186996; fax: +66 2 2186955; e-mail: twittie.s@chula.ac.th; corresponding author).

grouping) and compare the measurement values given by the BusCod model to determine which design (i.e., which grouping) better suits the component management goals that have been set.

This paper has the following organization. Section II discusses some related work. Section III presents our approach to apply the technique in [8] to a BPM of a particular application domain. A flight reservation system is the case study in Section IV and we give two designs of process components for this domain as an example. A discussion about the benefit of reusable process components is in Section V. Section VI presents the supporting tool for component measurement. The paper concludes and discusses future research directions in Section VII.

II. Related Work

Process-oriented software development tends to focus on the approaches to map BPMs to a process execution language for a particular software technology. Reference [1] addresses a correspondence between BPM and Web Services technology. The work in [9] realizes this approach by modeling a BPM with ADF or UML activity diagrams and transforming them to BPEL for execution. However such an approach attacks the implementation issue of the BPM, not the design and reuse issue.

A number of researches have addressed the concept of reusable business processes. In [6], a model for reusable business processes is proposed. A business process will be associated with information such as process function, process interface, and quality of service. Such information is for component cataloging and assembly purposes. In [7], a process pattern is used in the design of a software application but the work does not address how the pattern is designed.

For software component fabrication, the technique in [10] is close to the one in [8] which we will adopt. Software in [10] is also modeled using a UML class diagram and the model is decomposed according to different criteria such as data usage and business functions. Metrics based on cohesion and coupling of software components are proposed to measure the quality of the decomposition. However, the technique does not take managerial goals into account and consider less technical features than [8].

III. Measuring Process Component Design

We apply the technique in [8] to a BPM of an application domain as follows.

A. Design of Process Components

In [8], an application domain is modeled with a UML class diagram. The classes link with each other by association lines which represent relationships or connectivity that a class has with other classes. We can design software components by grouping together the classes that exhibit high degree of relationship or association with each other (e.g., Fig. 1). Analogously for a BPM such as a UML activity diagram in Fig. 2, the actions link with each other by control flow arrows. The arrows represent relationships or association between actions in terms of data coupling (i.e., data that are passed through) or time requirement (i.e., actions occur at the same instant of time) [10]. Therefore we may design process components by grouping together the actions that exhibit high degree of relationship with each other. Table I summarizes the correspondence between class diagram structure and BPM structure.

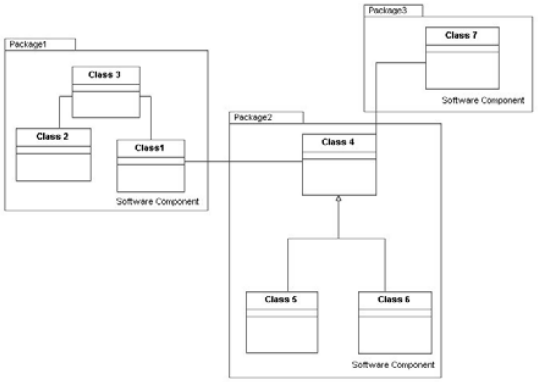

Fig. 1 Class model designed with three software components

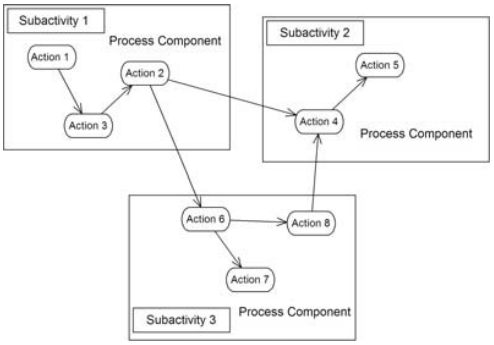

Fig. 2 Business process designed with three process components

TABLE I
Correspondence between structures of class diagram and BPM

Class diagram	BPM (activity diagram)
Class	Action
Link (inheritance or association)	Arrow (control flow)
Package	Subactivity

To design process components, a software designer will identify which part of the business process should be together as a process component for the domain. In Fig. 2, a software designer divides the actions in the activity diagram of the business process into three groups; each group is referred as a subactivity or a process component. Note that this is only one design for process components; the software designer can group the actions differently to create other designs.

B. Component Management Goals

Five component management goals for the design of software components [8] can be adopted for the design of

process components:

1) Cost Effectiveness (COST)

Cost effectiveness encompasses minimal component development cost and reduction in design and development time. This goal is important for achieving low cost business strategy. In component fabrication, costs depend on the actions included in the process component and the relationships among these actions.

2) Ease of Assembly (ASBL)

Ease of assembly refers to the ease with which components can be assembled. Reduction in the number of components required to assemble an application can enhance the assembly process since larger components will incorporate more functionality while complexities remain internal to the components. This goal is important for serving application developers who do not expect technical complexity at assembly.

3) Customization (CUST)

Customization is the ability to allow the application developers or assemblers to fit and alter solutions for a large variety of business applications using the components. This goal is important if the business competes in the market of customizable applications.

4) Reusability (REUS)

Reusability is the ability to reuse a component as is, without modification, in the development of various applications. Reusability also implies quality (i.e., conformance to requirements) and reliability (i.e., the ability to be depended on to correctly perform the function). The goal is important if the application developers are to be provided with components that can be used in many applications.

5) Maintainability (MNTN)

Maintainability is the ease with which the components can be added, deleted, or modified. This goal is important to low cost business strategy as maintenance may represent a long-term cost.

C. Technical Features

Five technical features for the design of software components [8] can be adapted for the design of process components:

1) Intercomponent Coupling (COUPL)

Intercomponent coupling is the strength of relationship between different components and low coupling is desired. In [8] where the application is designed with a UML class model, coupling is defined as the extent to which classes within a component relate in any way to other classes that are not in that component (e.g., by method invocation or by having the other classes as data types for attributes or method parameters of the class). For process components, actions in the activity diagram corresponds to classes in [8], and therefore coupling is defined as data coupling in terms of a control flow that carries data from one action in one component to the other action in the other component [11], or as time coupling by which two actions will occur together at an instant of time [10]. The measure for intercomponent coupling is as follows:

$$COUPL = \sum_{k=1}^{y} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n} (x_{ik} * (1 - x_{jk}) * c_{ij}) \quad (1)$$

where

y number of process components for the domain;
n total number of actions in the domain model;
x_{ik} 1 if action *i* is placed in process component *k*;
0 if action *i* is not placed in process component *k*;
c_{ij} coupling between action *i* and *j*, ($i,j \geq 0$; $i \neq j$).
Coupling between two actions can be measured from the number of control flow between them.

2) Intracomponent Cohesion (COHES)

Intracomponent cohesion is the strength of relationship within the component and high cohesion is desired. In [8], cohesion is defined as the extent to which classes within a component relate in any way to other classes within that component. For process components, cohesion is defined in terms of the control flow between actions of the same component. The measure for intracomponent cohesion is as follows:

$$COHES = \sum_{k=1}^{y} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n} ((x_{ik} * x_{jk}) * c_{ij}) \quad (2)$$

where

y number of process components for the domain;
n total number of actions in the domain model;
x_{ik} 1 if action *i* is placed in process component *k*;
0 if action *i* is not placed in process component *k*;
c_{ij} coupling between action *i* and *j*, ($i,j \geq 0$; $i \neq j$).

3) Number of Components (NCOMP)

Number of components represents the number of interface between components and complexity of that design, but a large number of components also give the application developers more choices in selecting the component that will closely satisfy the user requirement. The measure for number of components is as follows:

$$NCOMP = y \quad (3)$$

where

y number of process components for the domain.

4) Component Size (CSIZE)

Component size represents the granularity of the component set of the design. In [8], the measure for component size uses statistical standard deviation, instead of average size, to take into account the variability in the number of classes in different components. For process components, a similar normalized measure can be adopted as follows:

$$CSIZE = \sqrt{\sum_{j=1}^{y} \left(\sum_{i=1}^{n} x_{ij} \right)^2 / y} \quad (4)$$

where

y number of process components, $y > 0$;
n total number of actions, $n \geq 0$;

x_{ij} 1 if action *i* is placed in process component *j*;
0 if action *i* is not placed in process component *j*.

5) Complexity (COMPL)

The number of actions in a process component can provide a coarse-level complexity measure. In [8], complexity of a component is determined by the number of public methods and method parameters of classes within the component. Instead of a simple addition of the number of methods and parameter complexities across all components, the measure takes into account the variability of complexity between different designs by which the classes are distributed differently among the components. For process components, we may also adopt similar measurement. Since an action in the activity diagram corresponds to a UML class (c.f., Table I), we may look at each action as corresponding to a class with a single abstract operation which may take some input parameter data from the previous action in the process flow and produce some output parameter data to be passed onto the next action in the flow. This agrees with standard UML which allows a class method to associate with an action in an activity diagram [2]. However, the software designer will have to provide the details of such abstract operations for the business process. It is seen that the software designer can analyze the business process and can identify details (e.g., data that flow between actions). The measure is as follows:

$$COMPL = \sum_{j=1}^{y}\left(\sum_{i=1}^{n}\left(\left(w_{mcx} * m_i\right)+\left(w_{pcx} * \left(\sum_{k=1}^{m_i}\sum_{l=1}^{p_{ik}} p_{ikl}\right)\right)\right) * x_{ij}\right)^2 \quad (5)$$

where
y number of process components, $y > 0$;
n total number of actions, $n \geq 0$;
x_{ij} 1 if action *i* is placed in process component *j*;
0 if action *i* is not placed in process component *j*;
w_{mcx} relative importance of operation complexity, $(0 \geq w_{mcx} \leq 1)$;
m_i number of operation in action *i*, $(m_i = 1)$;
w_{pcx} relative importance of parameter complexity, $(0 \geq w_{pcx} \leq 1)$;
p_{ik} number of parameters in operation *k* in action *i*, $(p_{ik} \geq 0)$;
p_{ikl} relative complexity of the parameter *l* in operation *k* in action *i*, $(0 \geq p_{ikl} \leq 1)$.

The values for w_{mcx}, w_{pcx}, and p_{ikl} are subjective and assigned by the software designer during the design process. w_{mcx} and w_{pcx} can be assigned based on the complexity the software designer expects for the operations and parameters respectively. If the computation of the operations is likely to be complex in order to serve purpose of the actions, w_{mcx} may have high value. If it is necessary that the operations need a lot of complex parameters (e.g., those of complex data types) for their computation, then w_{pcx} may have high value. p_{ikl} is the degree of complexity of a parameter of an operation compared to that of other parameters of the same operation. A parameter is complex if, for example, it is of a complex data type.

By looking at an action as a class with an abstract operation, complexity measurement is refined. However, in most cases, an action in the activity diagram is modeled at high level and the software designer may find it inconvenient to analyze the details of the abstract operation and the complexity weights. In such cases, we propose a simplified formula for complexity based on the number of actions. This is analogous to the coarse-grained complexity measurement mentioned in [8] which is based on the number of classes in the class diagram. The simplified measure is as follows:

$$COMPL = \sum_{j=1}^{y}\left(\sum_{i=1}^{n} x_{ij}\right)^2 \quad (6)$$

where
y number of process components, $y > 0$;
n total number of actions, $n \geq 0$;
x_{ij} 1 if action *i* is placed in process component *j*;
0 if action *i* is not placed in process component *j*.

Table II is a summary of a literature survey on the impact that all the technical features may have on the component management goals [8]. Positive impact (+) means the higher the technical feature value is, the better the goal is achieved. Negative impact (-) means the lower the technical feature is, the better the goal is achieved. No impact (0) means the technical feature has no relation to achieving the goal.

TABLE II
IMPACT OF TECHNICAL FEATURES ON COMPONENT MANAGEMENT GOALS

	COUPL	COHES	NCOMP	CSIZE	COMPL
COST	0	-	-	-	-
ASBL	-	0	0	+	0
CUST	-	+	+	-	0
REUS	-	+	+	-	0
MNTN	-	-	-	-	-

D. Applying BusCod Model

We can adopt the BusCod model in [8]:

$$R' * W * D \quad (7)$$

Each part of the model is as follows.

1) $R' = \left[R_{COST} \quad R_{ASBL} \quad R_{CUST} \quad R_{REUS} \quad R_{MNTN}\right]$

This is a vector representing relative importance of all component management goals for the design where
R_{COST} relative importance of cost effectiveness;
R_{ASBL} relative importance of ease of assembly;
R_{CUST} relative importance of customization;
R_{REUS} relative importance of reusability;
R_{MNTN} relative importance of maintainability;
and $R_{COST} + R_{ASBL} + R_{CUST} + R_{REUS} + R_{MNTN} = 1$.

2)

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} \end{bmatrix}$$

This is a relation matrix representing the strength of association between managerial goals and technical features where

w_{ij} strength of the association between i^{th} managerial goal and j^{th} technical feature in W. The value is either 0 or 1-10 with the sign (+ or -) as in Table II.

3)

$$D = \begin{bmatrix} D_{COUPL} \\ D_{COHES} \\ D_{NCOMP} \\ D_{CSIZE} \\ D_{COMPL} \end{bmatrix}$$

This is a vector representing measurement of various technical features of the design where

D_{COUPL} intercomponent coupling;
D_{COHES} intracomponent cohesion;
D_{NCOMP} number of components;
D_{CSIZE} component size;
D_{COMPL} complexity.

E. Design Process

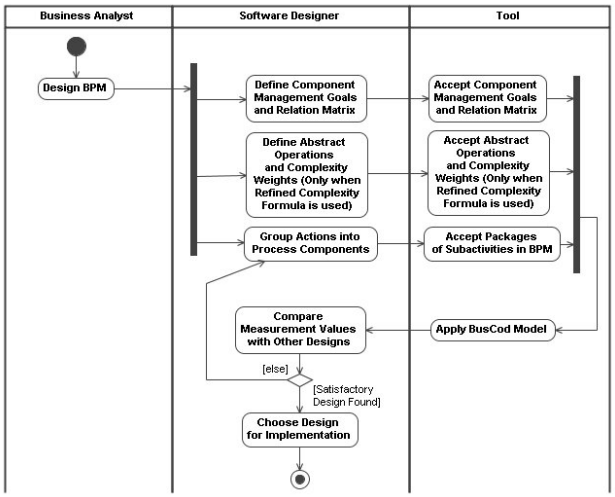

Fig. 3 Process for designing process components

Fig. 3 describes the process taken to design process components. This process will be supported by a software tool (see Section VI). The software designer obtains a BPM from a business analyst and defines relative importance of each managerial goal for the design. If the refined complexity formula is used, the software designer will also define details of abstract operations and complexity weights. Then, the BPM – a UML activity diagram in this case – will be decomposed into groups of actions; each group corresponds to a package. The packages will be processed by the tool to apply the BusCod model. The software designer can repeat this process with different designs and compare their measurement values from the BusCod model. The design with maximum measurement value will best achieve the managerial goals that have been set and can be chosen for implementation.

IV. Case Study

An example business process model used to demonstrate the design of process components is of the flight reservation domain, involving an airline (Fig. 4). We define this model by adapting from the Open Travel Alliance (OTA) specification [12] and the case study of [13]. The flow begins with checking a flight for a particular trip. Other details (i.e., schedule, arrival time, seat availability, and price) are checked subsequently. If seats are available and price is in budget, the flight is booked, the seat numbers are confirmed, and the itinerary is produced. In the case that no seats are available, this failed booking can be recorded for administrative purpose (e.g., to increase the number of flights during some period of the year).

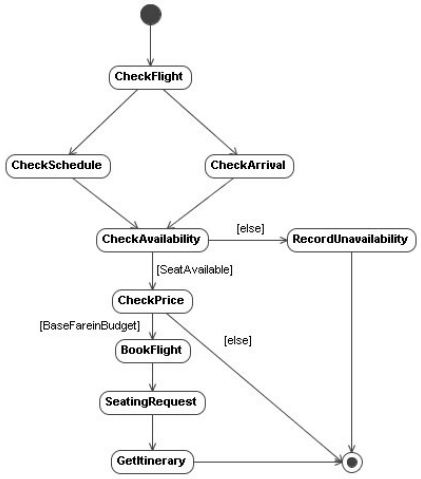

Fig. 4 Business process of flight reservation domain

According to the BusCod model, a software designer may want to design process components for this process flow with an emphasis on reusability. The component management goals may be set as

$$R' = \begin{bmatrix} 0.05 & 0.05 & 0.05 & 0.8 & 0.05 \end{bmatrix}$$

For this example, we use the matrix W below:

$$W = \begin{bmatrix} 0 & -8 & -6 & -8 & -7 \\ -5 & 0 & 0 & +8 & 0 \\ -6 & +5 & +5 & -5 & 0 \\ -8 & +8 & +6 & -7 & 0 \\ -7 & -6 & -5 & -6 & -8 \end{bmatrix}$$

This matrix is derived from a survey on the strength of relationship between the managerial goals and the technical features, conducted on industry experts [8].

Suppose the software designer decomposes the above process flow into three process components (Fig. 5). We may look at them as a flight inquiry component, a flight booking component, and a failed booking component. For later presentation purpose, we also annotate each action with a symbol A1-A9. This design then has its technical features calculated. We discuss the calculation of some values below.

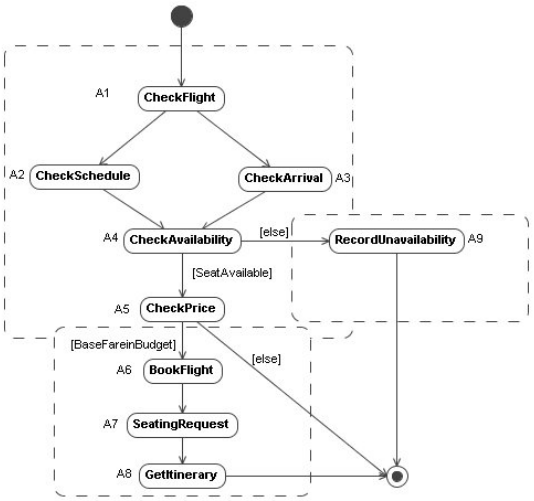

Fig. 5 A design with three process components

This design has 3 process components and 9 actions in total.

For intercomponent coupling and intracomponent cohesion, the value c_{ij} which is the coupling between any two actions in the process model is summarized in Table III.

TABLE III
COUPLING BETWEEN EACH ACTION

	A1	A2	A3	A4	A5	A6	A7	A8	A9
A1	0								
A2	1	0							
A3	1	0	0						
A4	0	1	1	0					
A5	0	0	0	1	0				
A6	0	0	0	0	1	0			
A7	0	0	0	0	0	1	0		
A8	0	0	0	0	0	0	1	0	
A9	0	0	0	1	0	0	0	0	0

Suppose the refined formula for complexity is used, the software designer specifies the operation detail for each action (Table IV). As mentioned in Section III.C, we can see each action as a single abstract operation. The design of the operations here is based on the technique to design service interfaces in [13] which considers elementary business functions and applies data normalization to interface parameters. This leads to minimization of coupling and maximization of cohesion of the operations.

In this example, the software designer also assigns a value 0.5 for the relative importance of operation complexity w_{mcx} and a value 0.5 for the relative importance of parameter complexity w_{pcx}. For each parameter in each operation, a value 1 is assigned for the relative complexity p_{ikl} as the parameters are equally complex (i.e., all are simple parameters).

The values of all technical features are:

$$D = \begin{bmatrix} 2 \\ 7 \\ 3 \\ 3.416 \\ 318.5 \end{bmatrix}$$

The calculation of the BusCod model $R^{'} * W * D$ for this three-component design gives the value -219.33. If we repeat the design process to calculate the BusCod value for a one-component design (i.e., the whole process flow is seen as one component), the BusCod value is -544.05. Therefore, the three-component design better achieves the component management goals that have been set. With an emphasis on the reusability goal, we may reason that the one-component design is less appropriate for reuse because it carries too much functionality.

TABLE IV
ACTIONS AS CLASSES WITH ABSTRACT OPERATIONS AND THEIR PARAMETERS

Abstract Operation of Action (No. of Parameters)	Input Parameters	Output Parameters
CheckFlight (4)	OriginalLocation DestinationLocation DepartureDate	FlightNumber
CheckSchedule (5)	FlightNumber	DepartureAirport DepartureTime ArrivalAirport ArrivalTime
CheckArrival (3)	FlightNumber DepartureDate	ArrivalDate
CheckAvailability (4)	FlightNumber DepartureDate CabinType	NoOfSeats
CheckPrice (6)	FlightNumber DepartureDate CabinType Budget	FareBasisCode BaseFare
BookFlight (5)	FlightNumber DepartureDate TravelerName CabinType	BookingReferenceID
SeatingRequest (3)	BookingReferenceID SeatPreference	SeatNumber
GetItinerary (12)	BookingReferenceID	BookingReferenceID FlightNumber DapartureAirport DepartureDate DepartureTime ArrivalAirport ArrivalDate ArrivalTime CabinType BookingStatus JourneyDuration
RecordUnavailability (3)	FlightNumber DepartureDate CabinType	

For this example, if the simplified formula is used for complexity, the calculation of the BusCod model for the three-component design gives the value -6.706 while the one-component design yields -58.05. The result still corresponds to the case where the refined formula is used.

V. Process Component Reuse

The quantitative measurement above can help give the software designer some confidence over the process component design. Process components from one business process model may be applied in the design of other business processes. For example, the flight inquiry component (top component) in Fig. 5 can be reused in the business process of another organization (Fig. 6). The organization views the reused flight inquiry process component as a single abstract action and composes it with organization's own process to additionally set booking information, buy and print flight tickets, and handle other unsuccessful booking incidents. Unlike the process in Fig. 5, ticket budget, seat assignment at the time of ticket purchase, and itinerary are not of concern to this business process.

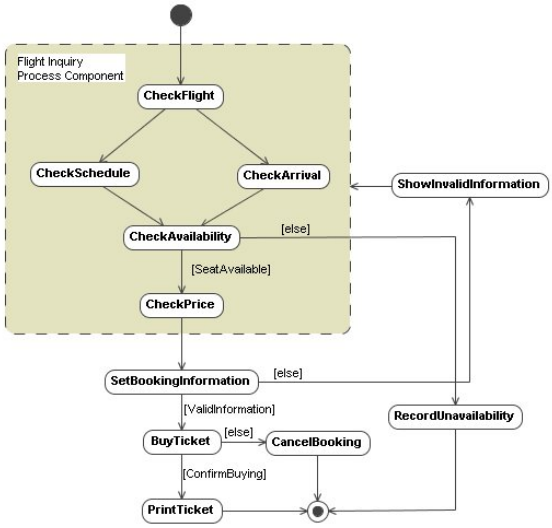

Fig. 6 Reusable process component in another business process

VI. Process Component Measuring Tool

A supporting tool has been developed to support the software designer to measure the quality of process components. According to Fig. 3, it is assumed that the business analyst will first use a modeling tool to design the activity diagram of the business process. The software designer will also use a modeling tool to define groups of subactivities. The result is an XMI-based file with packages of subactivities; the file will be an input to the tool.

In Fig. 7, the software designer can specify the input activity diagram file in the open file tab. Fig. 8 and Fig. 9 show respectively the managerial goals tab and the relation matrix tab where the software designer can fill in appropriate values. The measurement value tab in Fig. 10 calculates the BusCod value of the three-component model in Fig. 5 where a simplified complexity formula is used. The software designer will use this value to compare with those of other designs to determine the quality of the designs.

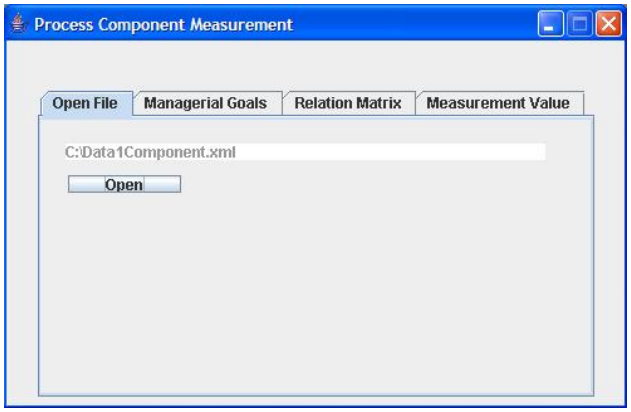

Fig. 7 Open file tab of the tool

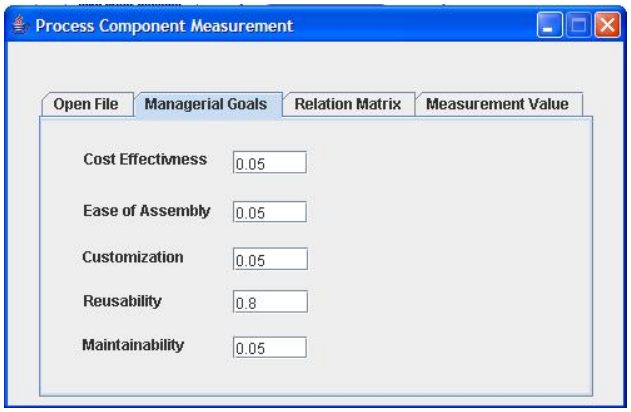

Fig. 8 Managerial goals tab of the tool

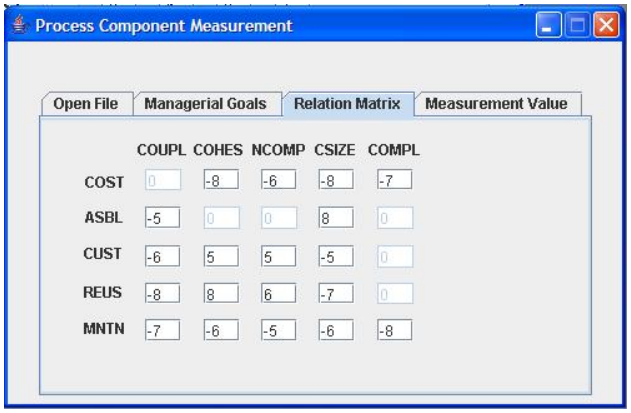

Fig. 9 Relation matrix tab of the tool

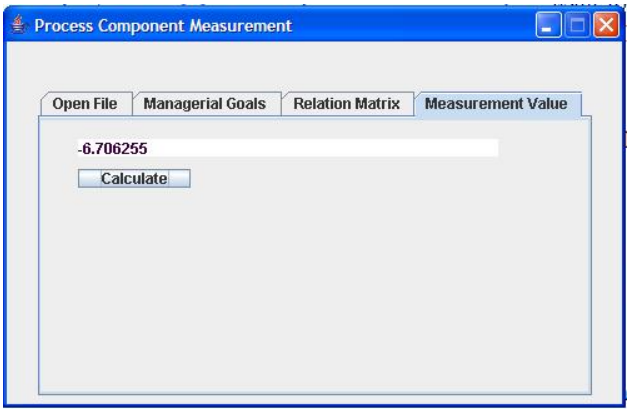

Fig. 10 Measurement value tab of the tool

VII. Conclusion

We discuss a possibility to apply a software component fabrication technique to design process components for an application domain which is modeled by a business process model. The paper relies on the BusCod model, the efficiency of which has been evaluated by industry experts as reported in [8]. Once a satisfactory design is found, the software designer can reuse each process component in the design of other business processes, and can have it implemented into a software unit. To implement a process component, we can follow the process-oriented paradigm and map each process component into BPEL [9], or we may follow the traditional paradigm and map each process component into a UML class diagram for component development [14].

This technique requires to a certain extent the skill of the software designer to group process components and to assign weight information for complexity. Other guidelines can help the software designer to decide which actions should be in the same process component (e.g., to reduce coupling and increase cohesion) [10].

The current version of the supporting tool can only give a BusCod value of a particular design. An enhancement is expected such that the tool can give an optimal design for a particular business process model.

Acknowledgment

The authors would like to thank Prof. Padmal Vitharana for guidance on the use of the BusCod model.

References

[1] BPMI.org. (2004, May, 3). *Business Process Modeling Notation (BPMN)* Version 1.0. Availalble: http://www.bpmi.org
[2] G. Booch, J. Rumbaugh, and I Jacobson, *The Unified Modeling Language User Guide*, Massachusetts: Addison-Wesley, 1999.
[3] E. Newcomer, Understanding Web Services. Indianapolis: Addison-Wesley, 2002.
[4] IBM. (2003, May, 5) *BPEL4WS Version 1.1 specification* Available: http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
[5] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd ed. New York: Addison-Wesley, 2002.
[6] Y. Mou, J. Cao, and S. Zhang, "A process component model for enterprise business knowledge reuse," in *Proc. IEEE International Conference on Services Computing (SCC04)*, 2004.
[7] O. H. Barros. (2004, September). *Business Information System Design Based on Process Pattern and Frameworks*. Industrial Engineering Department, University of Chile. Available: http://www.BPtrends.com
[8] P. Vitharana, H. Jain, and F. M. Zahedi, "Strategy-based design of reusable business components," *IEEE Transactions on Systems, Man, and Cybernetics–Part C: Applications and Reviews*, vol. 34, No. 4, pp. 460-474, November 2004.
[9] J. Koehler, R. Hauser, S. Kapoor, F. Y. Wu, and S. Kumaran, "A model-driven transformation method," in *Proc. 7th IEEE International Enterprise Distributed Object Computing Conference*, 16-19 September 2003, pp. 186-197.
[10] N. Tagoug, "Object-oriented system decomposition quality", in *Proc. 7th International Symposium on High Assurance Systems Engineering (HASE02)*, 2002.
[11] B. Husslage, E. van Dam, D. den Hertog, P. Stehouwer, and E. Stinstra, *Coordination of coupled black box simulations in the construction of metamodels*, Discussion Paper 2, Center for Economic Research, Tilburg University, 2003.
[12] Open Travel Alliance (OTA). (2005, December, 05). *OTA Specification 2005B*. Available: http://www.opentravel.org
[13] G. Feuerlicht and S. Meesathit, "Design framework for interoperable service interfaces," in *Proc. 2nd International Conference on Service Oriented Computing (ICSOC'04)*, New York, 2004, pp. 299-307.
[14] W. Rungworawut and T. Senivongse, "A guideline to mapping business process to UML class diagrams," *WSEAS Transactions on Computers*, vol. 4, November 2005.