Distance Measures for Information System Reengineering

Geert Poels, Stijn Viaene, and Guido Dedene

Management Information Systems Group
Department of Applied Economic Sciences
Katholieke Universiteit Leuven
Naamsestraat 69, B-3000 Leuven, Belgium
{geert.poels, stijn.viaene, guido.dedene}@econ.kuleuven.ac.be

Abstract. We present an approach to assess the magnitude and impact of information system reengineering caused by business process change. This approach is based on two concepts: object-oriented business modeling and distance measurement. The former concept is used to visualize changes in the business layer of an information system architecture. The latter concept is used to quantify these changes. The paper also describes the application of our approach in the context of front-office system design.

1 Introduction

There exists a wide spectrum of reasons for information system reengineering. The evolution of an information system over many years frequently leads to software that is unnecessarily complex and inflexible, making the maintenance and enhancement of the system more and more expensive [1]. Often, legacy systems are reengineered because of a desire to move to new generations of software technology, like component software [2]. Changes in the computer and network infrastructure may as well trigger reengineering efforts.

The focus of this paper is however on a different type of reengineering, i.e. information system reengineering because of changing business processes. Major strategic management decisions regarding business re-positioning and drastic business transformations (i.e. BPR as defined in [3], [4]) require the current business process(es) to be changed and the existing information system to be modified in order to further support the business operations. A characteristic of this type of information system reengineering is that it is fundamental. It affects the enterprise model, which is the core layer in an object-oriented information system architecture [5]. The enterprise model is an abstract image of business reality (either a single business process or a network of interrelated processes), capturing the relevant business entities, events and rules, their static relationships and dynamic interactions. In the implemented software system, the classes of the enterprise model are responsible for the business functionality that is offered by the information system.

Reengineering the information system is crucial for the overall success of a business process change. One aspect to consider is the reengineering cost. Promised benefits must be balanced against this cost, preferably before the actual change(s) take(s) place. In this paper we present an approach to assess the magnitude and

B. Wangler, L. Bergman (Eds.): CAiSE 2000, LNCS 1789, pp. 387-400, 2000

impact of information system reengineering, more precisely the amount of changes in the enterprise model. An assessment of the effect of 'business process change'-driven reengineering on the core layer of an information system provides complementary quantitative information (apart from financial measures like ROI, etc.) to support strategic decision making with respect to business re-positioning and business transformation. Measurement allows assessing how 'big' the changes are, and thus provides an objective means to compare alternatives.

Our approach is based on two concepts: business modeling and distance measurement. First, a conceptual modeling method, called object-oriented business modeling by contract [6], is used to model the AS-IS enterprise model (i.e. the model before the business process change(s) take(s) place) and the TO-BE enterprise model (i.e. the model as it would look like after the business process change(s)). Comparing these models allows visualizing the changes that would be caused by reengineering. Next, the distance between the models is measured to quantify the magnitude of the changes. The basic assumption underlying our approach is that the larger the distance between the AS-IS and TO-BE models, the larger the amount of changes that must be handled, and thus the larger the impact and cost of information system reengineering.

This paper is organized as follows. Section 2 discusses previous work on distance measurement in the context of software reengineering. Section 3 presents the modeling and measurement aspects of our approach. In section 4 we discuss some experiences in using our approach in the context of front-office system design. Conclusions and topics for further research are presented in section 5.

2 Related Work

Distance measurement is of course not a new concept in the software reengineering field. Such measurements have been used to cluster components of a software system into subsystems or modules, for instance, to improve the modularity of the system [7], or to reverse engineer the system (i.e. design recovery) [8]. They have also been used to assess the cohesion of subsystems [9], again with the purpose of reengineering the system. In other software engineering related fields, distance measurement has been proposed as a technique for component reusability assessment [10] and component retrieval [11].

Generally, these approaches use distance measurement to assess the closeness between two software entities in terms of the properties they have in common and the properties that are unique to each of the entities. Most measures of 'closeness' take the form of a normalized similarity (or affinity) measure. Given a set of software entities E (e.g. classes in an object-oriented system), a set of properties P (e.g. the union of class methods in the system), and a function p mapping entities of E into subsets of P (e.g. all methods defined in a class or used by the class), the degree of similarity between $A \in E$ and $B \in E$ with respect to P is measured by

$$Sim(A,B) = \frac{|p(A) \cap p(B)|}{|p(A) \cup p(B)|} \quad . \tag{1}$$

This measure returns the relative amount of common properties between A and B. The higher this value, the more similar A and B are with respect to P.

It is obvious that *Sim* works fine for purposes like cluster analysis and component matching. However, we need a measure for the distance (and not the closeness) between two software entities. In other words, we are not interested in the properties that A and B have in common, but in the properties in which they are different.

A measure that puts more emphasis on the difference between A and B is the dissimilarity measure *Dis* as proposed, for instance, in [9]

$$Dis(A,B) = \frac{\left|p(A) \cup p(B)\right| - \left|p(A) \cap p(B)\right|}{\left|p(A) \cup p(B)\right|} \quad . \tag{2}$$

This measure however does not express the magnitude of the dissimilarity between A and B. The measure is normalized and thus implicitly accounts for the relative amount of common properties between A and B. For our research purpose we need to quantify the absolute amount of differences between entities A and B, regardless of how many properties they have in common.

An alternative measure is the distance measure *Dist*

$$Dist(A,B) = \left|p(A) \cup p(B)\right| - \left|p(A) \cap p(B)\right| = \left|p(A) - p(B)\right| + \left|p(B) - p(A)\right| \quad . \tag{3}$$

This measure returns the cardinality of the symmetric difference between p(A) and p(B). It focuses upon the changes that are needed to turn entity A into entity B, and vice versa.

Our approach to distance measurement uses measures of the form *Dist*. The approach is different from previous research in the sense that distance measures satisfy the metric axioms [12]. Working with metrics offers the additional advantage that the measures can be formally validated within the framework of measurement theory, as required by software measurement scientists [13], [14], [15].

3 A Distance-Based Change Assessment Approach

We first present a brief introduction to object-oriented business modeling by contract. Next, a generic method to define distance measures for software entities is presented and illustrated for OO business models.

3.1 OO Business Modeling by Contract

Object orientation has proven an excellent paradigm to model business processes [16], [17]. Object-oriented enterprise models improve the communication between business professionals and software engineers. Moreover, many OOAD methods (see e.g. [18], [19]) develop information systems starting from such models. The enterprise model does not only describe the functioning of a business process; it is an integral part of the specifications of the information system that supports the business process. The object classes, identified during business modeling, take a prominent place amongst the classes in the OO information system, once it is implemented. Therefore, a comparison of the AS-IS and TO-BE enterprise models provides insight

into business process changes as well as information system changes. Moreover, this insight is gained before the actual changes take place.

The object-oriented business modeling by contract method [6] places much emphasis on the concept of a 'business event', in so far as it has been classified as an event-driven method [20]. At the highest level of abstraction, a business process is seen as a sequence of occurrences of business events. The business entities affected by these occurrences are modeled as business objects. Business objects and events are further classified into types and subtypes (i.e. generalization / specialization).

Mathematically, each business object type is defined through the set of business event types it is involved in. Object life cycle models are used to specify sequence constraints on the participation of business objects into business events. They thus model an important class of business rules. The communication and synchronization between objects is modeled by means of common event participation (i.e. event broadcasting instead of message passing). This principle also allows modeling well-known structural relationships between business object types (e.g. categorization, aggregation, composition, association) in terms of existence dependency associations and/or contract object types.[1] The main advantage of the existence dependency concept is that the consistency between the static and dynamic aspects of the business model can be formally verified.[2]

At lower levels of abstraction, business object types are given a class definition, which can be gradually refined throughout the development process. Generalization / specialization relationships between business object types are specified by means of inheritance relationships between the corresponding classes. Existence dependency associations are implemented through the data abstraction mechanism (e.g. by means of attributes that are used as pointers to existence dependent and/or master classes). The end result is a fully (and formally) specified object-oriented model of the business process that acts at the same time as the kernel of a layered architecture for the information system [22].

As an illustration, part of the specification for a (simplified) library's loan circulation process is shown below. Fig. 1 presents the class diagram in UML notation [23]. Table 1 is an object-event association matrix, showing which business object types / classes are affected by the occurrence of business events. The symbols "C", "M", and "E" indicate respectively that the occurrence of the business event (of the type shown in the row header) creates, modifies or ends the life of a business object (of the type shown in the column header).

---

[1] A business object *x* is existence dependent on a business object *y* if *x* is during its life always associated to *y* (i.e. *x* cannot be created before *y* and it can no longer exist when the life of *y* has ended). In a library for instance, each LOAN object is always associated to a particular BOOK object. Therefore we say that the object type LOAN is existence dependent on the object type BOOK. For the existence dependent object type (e.g. LOAN), the association with the master object type (e.g. BOOK) is mandatory with a connectivity (cardinality, multiplicity) of one.

[2] The semantic integrity between the static business model (e.g. class diagram) and the dynamic business model (e.g. object-event association matrix) is guaranteed when the existence dependent object type is a subset of the master object type. Note that at this level of abstraction business object types are defined as sets of business event types. For a formal proof we refer to [21].

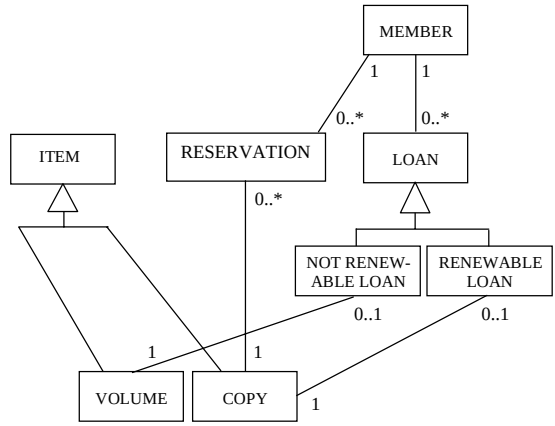

Fig. 1. Class diagram of a library

3.2 Distance Measures for OO Business Models

Our approach to define distance measures for software entities has been fully described elsewhere [24]. Here, we just present the basic principles.

Two software entities can be different in many aspects. For each aspect that is considered, a model or abstraction of the software entity is defined. The distance between software entities A and B with respect to aspect X is then measured by the count of elementary transformations (i.e. atomic changes of a given type) that are minimally needed to transform the model of A (for X) into the model of B (for X). The more elementary transformations that are needed, the larger the distance between A and B, with respect to aspect X.

As an example, the distance between business object types VOLUME and COPY can be expressed in terms of their involvement in business event types (cf. Table 1). The relevant model of an object type is here the set of event types it is involved in (i.e. all event types having their entries marked in the column for the object type in the object-event association matrix). Elementary transformations are of the type 'adding an event type' or 'removing an event type'. It can easily be seen that for VOLUME and COPY it takes minimally 12 such transformations. Hence, their distance with respect to business event type involvement is 12.

The measure thus defined is of the form *Dist* as presented in section 2. The aspect of distance X, i.e. business event type involvement, corresponds here to the set of properties P in the general definition of *Dist*. The function p referred to in this definition, maps the object types A and B into their mathematical definition. Hence, the sets p(A) - p(B) and p(B) - p(A) contain the business event types that have to be added or removed by means of elementary transformations. Consequently, the sum of the cardinalities of these sets equals the minimum number of elementary transformations that are needed to transform object type A into object type B, or vice versa.

Table 1. Object-event association matrix for a library

	ITEM	VOLUME	COPY	RESERVATION	MEMBER	LOAN	NOT_RENEWABLE_LOAN	RENEWABLE_LOAN
Acquire	C							
acq_vol.		C						
acq_cop.			C					
Catalogue	M	M	M					
Sell	E							
sell_vol.		E						
sell_cop.			E					
Reserve			M	C	M			
Cancel			M	E	M			
Fetch			M	E	M			C
Register					C			
Leave					E			
Borrow					M	C		
cr_nrloan		M			M		C	
cr_rloan			M		M			C
Return		M	M		M	E	E	E
Lose					M	E		
Lose_vol.		E			M		E	
Lose_cop			E		M			E
Renew			M		M			M

We have defined sets of elementary transformation types for different kinds of models like sets, multi-sets, matrices, state machines, etc. Similar notions for distance between trees, strings, clusters, etc. can be found in the literature [25], [26], [27]. An important result of our research is that this particular way of defining a distance measure results in a function that satisfies the metric axioms. The good news is that metric functions fit into the framework of measurement theory. According to [12], metrics are homomorphic mappings of proximity structures into metric spaces, i.e. they map an empirical notion of distance into a mathematical notion of distance. Another result from measurement theory is that metrics define ordinal scales of distance. They allow distance values to be ranked, which is useful when comparing alternatives.

In [28] we further showed that distance measures based on counts of elementary transformations represent a special type of proximity structure, i.e. segmentally additive proximity structures. Such a representation results in the definition of ratio scales of distance, which allows expressing distance values in ratios and percentages [12].

It should be noted that metrics focus on the difference between two entities, without regard to how much these entities have in common. We believe such a point of view is justified for the type of application described in this paper: to quantify the *amount of change* in the enterprise model due to business process changes.

4 Applying the Approach to a Reference Framework for Front-Office System Design

We applied our approach to a reference framework for front-office system design [29]. This framework concerns the organization of the front-office, i.e. the part of a service organization where the services required by a customer and offered by the service provider are agreed upon. The framework is based on the concept of service customization, as proposed by the management scientists Lampel and Mintzberg [30]. It distinguishes five types of front-office depending on the level of service customization: pure standardization, segmented standardization, customized standardization, tailored customization, and pure customization. Each type of front-office requires its own specific information system to support its specific information requirements. The framework proposes an object-oriented business model for each type of front-office. These models can be used as reference models for actual front-office system design.

The framework of de Vries is useful for companies wishing to introduce a front-office organization and its supporting information system. It is also useful as a strategic management instrument for changing the service specification process. Companies wishing to move to higher levels of service customization can use the framework to reengineer their front-office system. However, before such a move is decided on, companies must have an idea of the impact of the reengineering. This is a question that can be addressed by our distance-based change assessment approach.

In a first sub-section we present the generic front-office enterprise models of de Vries. The models in [29] were already specified using the object-oriented business modeling by contract method. For the sake of brevity, we only present here the static business models, i.e. class diagrams in UML notation. In the next sub-section we propose a distance measure for UML class diagrams. In a final sub-section the measurement results are presented and analyzed.

4.1 Generic Front-Office Object Models

Overall, the front-office needs product-information, process-information and information on the customer-relationship. Table 2 shows for each level of service customization and corresponding front-office type the type of information that is needed.

Table 2. Information model of front-office customer interaction of de Vries [29]

FRONT-OFFICE TYPE	DEGREE OF CUSTOM-IZATION	RELATION-RELATED	PRODUCT-RELATED	PROCESS-RELATED
Counter	Pure standardization	Anonymous transactions	End products	Delivery times for products
One stop shop	Segmented standardization	Characteristics of market segments	Assortments	Delivery times for assortments
Field and inside service	Customized standardization	Customer profiles	Standard components	Available capacity
Control room	Tailored customization	Development of the relationship	Smallest replicating unit	Capacity assignment
Symbiosis	Pure customization	Opportunities for partnership	Design knowledge	Implementation and outsourcing opportunities

Fig. 2 presents the 'counter' model. The information needed for standardized service transactions can be encapsulated in the SUPPLIER, SERVICE and TRANSACTION classes in the diagram. For instance, descriptive attributes of the SERVICE class include the service functionality description, price and warranty conditions and service procedure descriptions. Transaction amounts and timestamps are descriptive attributes of the TRANSACTION class.

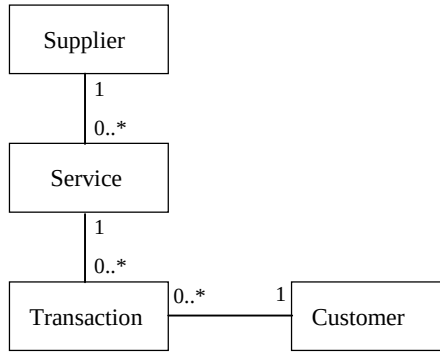

Fig. 2. The counter model

A 'one-stop-shop' offers a specific assortment of services to customers depending on the market segment to which they belong (Fig. 3). The essential front-office processes are the determination of market segments and assortments. Information like segmentation criteria, assortment discount rates, etc. can readily be encapsulated in the front-office enterprise classes.

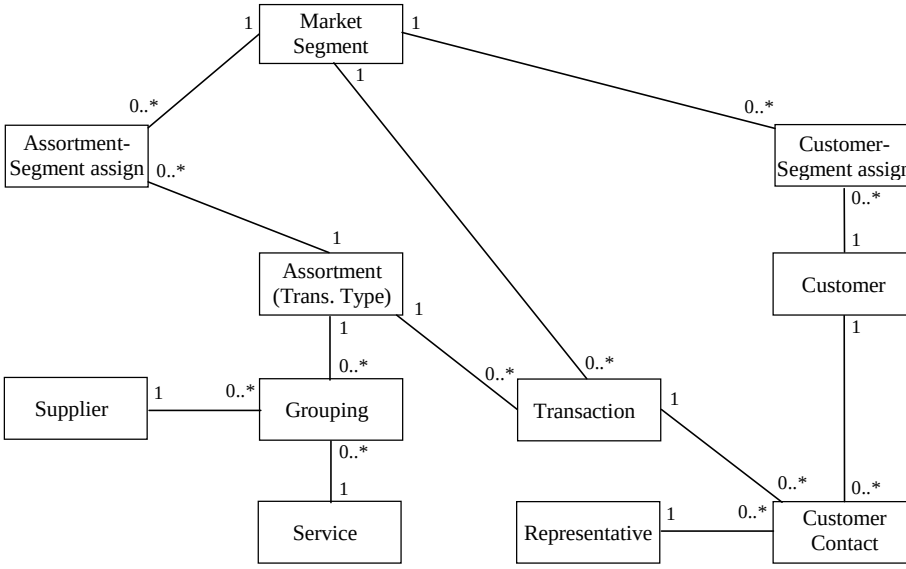

Fig. 3. The one-stop-shop model

According to de Vries *et al.* the 'one-stop-shop' model unfolds from the 'counter' model. The primary effects of customization are the concepts of segmentation and assortments.

The 'field and inside service' type of front-office is a clear extension of the 'one-stop-shop' (Fig. 4). To offer customized standardization the front-office is organized into a field service (e.g. sales people) that is supported by an inside service. The former is responsible for the customer relationships, whereas the latter is responsible for profiling, matching, and the bundling and offering of services.

Figs. 5 and 6 show the models of the 'control room' and 'symbiosis' types of front-office. The models clearly show that the emphasis shifts from product and process related information to the customer relation. The 'control room' front-office aims to establish a structural link with commercially attractive customers by means of tailored customization. Within the bounds of the standard service design and delivery process, the front-office representative and the customer specify the service to be provided. In the 'symbiosis' model the service provider and the customer collaborate completely in the various steps of designing, acquiring, and producing customized services.

4.2 A Distance Measure

A distance measure is needed for the UML class diagrams of Figs. 2 to 6. Note that these diagrams are built from only two types of elements: classes and existence dependency associations between classes. Note also that all associations are characterised by the same connectivity constraints (i.e. mandatory with a connectivity of 1 on one side and optional with a connectivity of many on the other side).

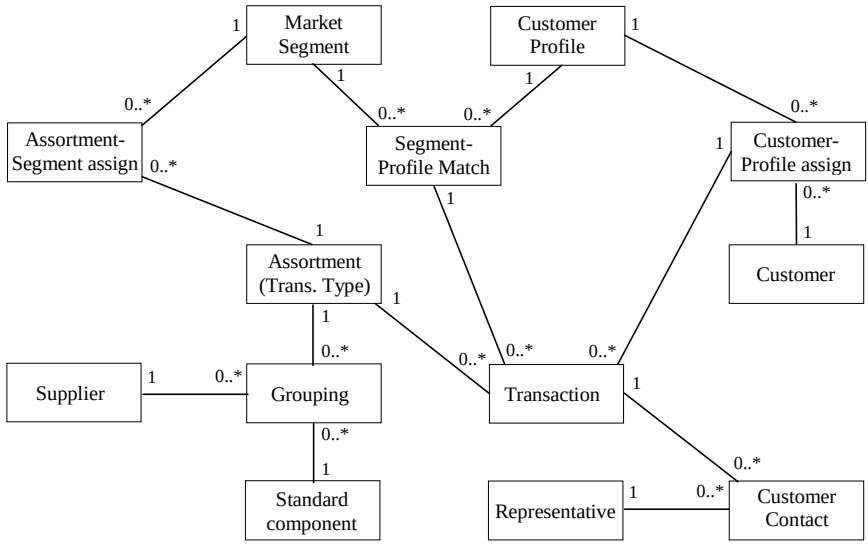

Fig. 4. The field and inside service model

A set of elementary transformation types that is sufficient to express all different types of atomic change in the UML class diagrams considered here, is the following:

t_1: remove a class from the diagram;
t_2: add a class to the diagram;
t_3: remove an association;
t_4: add an association.

The distance between any pair of front-office types, with respect to their static business models (i.e. the UML class diagrams of Figs. 2 to 6), is measured by the minimum number of elementary transformations of the types t_1 to t_4 that are needed to transform one model into the other.

The application of a distance measure for software entities may require syntactic, semantic and/or linguistic rules to decide on two entities being identical or not. We assume here that strict class labeling conventions have been followed, such that classes with the same name can be considered as identical. By convention, if part of the class name is between brackets, then only this part is used for matching classes. Associations are identical across models if the participating classes are also identical: if classes C and C' are associated in model M_i, and class C' is replaced by class C" in model M_j, without changing the connectivity constraints of the association with C, then we also consider this association to be the same in M_i and M_j. Generally however, removing a class implies that all the associations it is involved in are removed as well.

4.3 Analysis and Discussion of Measurement Results

Table 3 shows the distance measurements between all possible pairs of generic front-office object model. As an example, consider the distance between the 'control room'

model (Fig. 5) and the 'symbiosis' model (Fig. 6). The table shows that 12 atomic changes are needed to transform one model into the other, or vice versa. Starting from the 'control room', the classes STANDARD COMPONENT, STANDARD SERVICE, TRANSACTION, REPRESENTATIVE and CUSTOMER CONTACT, and the associations STANDARD SERVICE - GROUPING, STANDARD SERVICE - TRANSACTION, TRANSACTION - CUSTOMER CONTACT and REPRESENTATIVE - CUSTOMER CONTACT are removed, whereas the classes RESOURCE and ACTION, and the association ACTION - GROUPING are added to obtain the 'symbiosis' front-office model.

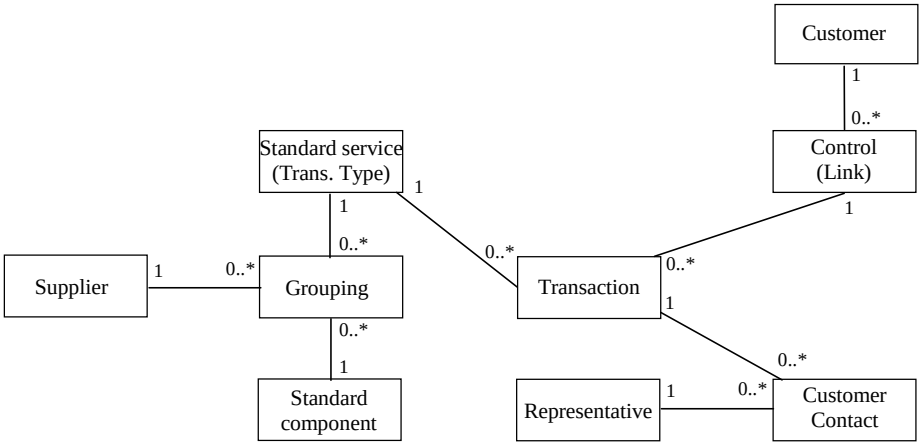

Fig. 5. The control room model

Table 3 can be used as a complementary evaluation instrument by companies wishing to reengineer their front-office and supporting front-office information system, in order to move to another level of service customization. Suppose for instance that a service organization with a 'counter' office wishes to offer customized services. The values of Table 3 suggest that a move towards segmented or customized standardization requires a far greater impact on the front-office enterprise model than a move towards the highest levels of service customization. The values also suggest that companies that gradually move towards the highest levels of service customization will face more changes in the beginning of this process than in the end. Note further the effect of the 'triangle inequality', i.e. one of the metric axioms. For instance, whereas the distance between the 'one stop shop' and the 'control room' is 14 and the distance between the 'control room' and the 'symbiosis' model is 12, the distance between the 'one stop shop' and the 'symbiosis' model is only 24, i.e. less than 14 + 12. The 'strict' triangle inequalities observed in Table 3 strongly suggest that it might be sub-optimal to move the level of service customization one step at a time. A more drastic reengineering of the current front-office may pay off in the long term.

The values in Table 3 must of course be interpreted with care. Moving to another level of service customization and re-organizing the front-office requires more than changing the front-office information system. Besides, the values only reflect the amount of change required for the enterprise model layer in the architecture of the front-office system. In the absence of further (empirical) studies, we can only assume

that the impact on the other layers of the system architecture is proportional to the amount of enterprise model changes. The same remark holds for the reengineering costs. It is for instance assumed that each type of elementary transformation involves the same reengineering cost, which is of course only an approximation of reality. Finally, note that the generic object models of Figs. 2 to 6 are to be seen as domain models, that must be instantiated for individual companies. The actual front-office enterprise models might thus be different from the domain models proposed in the reference framework. As a consequence, the distance values might be different too.

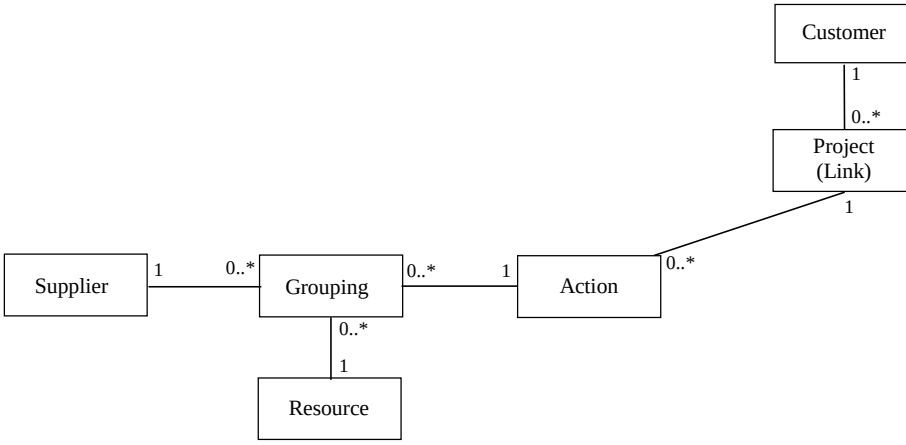

Fig. 6. The symbiosis model

Table 3. Distance values for the generic front-office object models in the reference framework of de Vries

	counter	one stop shop	field and inside service	control room	symbiosis
counter	0	22	28	18	14
one stop shop	22	0	16	14	24
field and inside service	28	16	0	16	28
control room	18	16	16	0	12
symbiosis	14	24	28	12	0

5 Conclusions and Further Research Topics

The approach presented in this paper provides information regarding the impact and magnitude of information system reengineering caused by business process change. Measuring the distance between the AS-IS and TO-BE models of a business process helps quantifying the amount of change that is needed to reengineer the supporting information system. We must note that further research, mainly empirical in nature, is needed to relate this modeled and measured amount of change to management variables like reengineering costs, migration costs, and risks (e.g. potential data loss).

We also acknowledge that measuring static enterprise models, as in section 4, gives only one view on the complex problem of 'business process change'-driven information system reengineering. A balanced approach requires measuring a whole array of static and dynamic product models, as well as process models, workflow models, etc.

Our distance-based modeling and measurement approach can be applied in other contexts too. In [24] a method is proposed to measure software attributes (e.g. coupling, cohesion, size) in terms of distances between software product models, that emphasize such attributes, and 'reference' models, that represent 'ideal' models for the attributes. In [31] this method has been used to measure the reuse of object-oriented business models. A topic of future research is to use distance measurement in object-oriented business models for the identification of reusable business (software) components [32]. In our opinion and experience, the concepts of distance and metric are both flexible and formal, allowing them to be used in a variety of software (re)engineering contexts.

Acknowledgements

Geert Poels is Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium)(F.W.O.). Stijn Viaene holds the KBC Insurance Research Chair in Management Information Systems at the Katholieke Universiteit Leuven. We wish to thank the anonymous referees for their valuable comments.

References

1. Ciupke, O.: Automatic Detection of Design Problems in Object-Oriented Reengineering. In: Proc. TOOLS'99. Santa Barbara, Calif. (1999) 18-32
2. Sahraoui, H.A., Melo, W., Lounis, H., Dumont, F.: Applying Concept Formation Methods to Object Identification in Procedural Code. In: Proc. 12th Int'l Automated Software Eng. Conf. (ASEC'97). Incline Village, Nev. (1997) 210-218
3. Davenport, T.H., Short, J.E.: The New Industrial Engineering: Information Technology and Business Process Redesign. Sloan Management Review. 31 (1990) 11-27
4. Hammer, M., Champy, J.: Reengineering the Corporation, A Manifesto for Business Revolution. Harper, New York (1993)
5. Zachman, J.A.: A framework for information architecture. IBM Systems J. 26 (1987) 276-292
6. Snoeck, M., Dedene, G., Verhelst, M., Depuydt, A.: Object-Oriented Enterprise Modelling with MERODE. University Press, Leuven (1999)
7. Brito e Abreu, F., Peirera, G., Sousa, P.: Reengineering the Modularity of Object-Oriented Systems. In: Proc. ECOOP'98, Workshop Reader, Workshop on Techniques, Tools and Formalisms for Capturing and Assessing the Architectural Quality in Object-Oriented Software. Brussels (1998) 62-63
8. Tzerpos, V., Holt, R.C.: Software Botryology: Automatic Clustering of Software Systems. In: Proc. Int'l Workshop on Large Scale Software Composition. Vienna (1998)
9. Simon, F., Löffler, S., Lewerentz, C.: Distance Based Cohesion Measuring. In: Proc. 2nd European Software Measurement Conf. Amsterdam (1999) 69-83
10. Castano, S., De Antonellis, V., Zonta, B.: Classifying and Reusing Conceptual Schemas. In: Proc. 11th Int'l Conf. on Conceptual Modelling (ER'92). Karlsruhe (1992) 121-138

11.Jilani, L.L., Mili, R., Mili, A.: Approximate Component Retrieval: An Academic Exercise or a Practical Concern? In: Proc. 8th Workshop on Institutionalising Software Reuse (WISR8). Columbus, Ohio (1997)
12.Suppes, P., Krantz, D.M., Luce, R.D., Tversky, A.: Foundations of Measurement: Geometrical, Threshold, and Probabilistic Representations. Academic Press, San Diego, Calif. (1989)
13.Briand, L., El Emam, K., Morasca, S.: Theoretical and Empirical Validation of Software Product Measures. Tech. rep. ISERN-95-03, International Software Engineering Network (1995)
14.Fenton, N., Pfleeger, S.L.,: Software Metrics: A Rigorous and Practical Approach. International Thomson Computer Press, London (1997)
15.Zuse, H.: A Framework for Software Measurement. Walter de Gruyter, Berlin (1998)
16.Wang, S.: OO Modeling of Business Processes: Object-Oriented Systems Analysis. Information Systems Management. (1994) 36-43
17.Dedene, G., Snoeck, M.: Generic Object Models and Business Process (Re)Design. Tech. rep. DTEW 9667, Catholic University of Leuven (1996)
18.Jacobson, I., et al.: Object-Oriented Software Engineering, A Use Case Driven Approach. Addison-Wesley, Reading, Mass. (1992)
19.D'Souza, D.F., Wills, A.C.: Objects, Components, and Frameworks with UML: The Catalysis Approach. Addison-Wesley, Reading, Mass. (1998)
20.Simons, A.J.H., Snoeck, M., Hung, K.S.Y.: Design Patterns as Litmus Paper to Test the Strength of Object-Oriented Methods. In: Proc 5th Int'l Conf. on Object Oriented Information Systems (OOIS'98). Paris (1998)
21.Snoeck, M., Dedene, G.: Existence Dependency: the key to semantic integrity between structural and behavioural aspects of object types. IEEE Trans. Software Eng. 24 (1998) 233-251
22.Poels, G.: Evaluating the Modularity of Model-Driven Object-Oriented Software Architectures, In: Proc. ECOOP'98, Workshop Reader, Workshop on Techniques, Tools and Formalisms for Capturing and Assessing the Architectural Quality in Object-Oriented Software. Brussels (1998) 52-53
23.Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, Reading, Mass. (1999)
24.Poels, G., Dedene, G.: Distance-based software measurement: necessary and sufficient properties for software measures. Information and Software Technology. 42 (2000) 35-46
25.Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. Siam J. on Computing. 18 (1989) 1245-1262
26.Oommen, B.J., Zhang, K., Lee, W.: Numerical Similarity and Dissimilarity Measures Between Two Trees. IEEE Trans. on Computers, 45 (1996) 1426-1434
27.Tzerpos, V., Holt, R.C.: MoJo: A Distance Metric for Software Clusterings, In: Proc. 6th Working Conf. on Reverse Engineering (WCRE'99). Atlanta (1999)
28.Poels, G., Dedene, G.: Modelling and measuring object-oriented software attributes with proximity structures, In: Proc. 3rd Int'l Workshop on Quantitative Approaches in Object-Oriented Software Eng. Lisbon (1999) 1-22
29.de Vries, E.J., Maes, R., Dedene, G., Viaene, S., Poels, G., Snoeck, M.: Object Models for Customer Relations in the Front-Office. Tech. rep. PrimaVera 98-11, University of Amsterdam (1998)
30.Lampel, J., Mintzberg, H.: Customizing Customization, Sloan Management Review. 38 (1996)
31.Snoeck, M., Poels, G., Dedene, G.: Reusing Business Models. Tech. rep. DTEW 9934, Catholic University of Leuven (1999)
32.Poels, G., Dedene, G.: Moving from OOAD to COAD. In: Proc. 8th Object Technology Conf. (OT'2000). Oxford (2000)