A Framework for Visualization of Changes of Enterprise Architecture

Robert Bakelaar[1], Ella Roubtsova[2(✉)], and Stef Joosten[2]

[1] Royal Vopak, Global IT, Westerlaan 10, 3016CK Rotterdam, The Netherlands
[2] Open University of the Netherlands, Valkenburgerweg 177,
6401DL Heerlen, The Netherlands
Ella.Roubtsova@ou.nl

Abstract. An innovation that is substantial enough to change the enterprise architecture poses a problem for a system architect. Enterprise architecture modeling methods and tools do not support the distinction between the As-Is architecture and the To-Be architecture in one view model. Recognizing the changes becomes similar to a game of "finding changes in two drawings". As the size of architectures and number of architectural view pairs grows, the changes can be overlooked. In order to support the recognizing of changes by the implementation teams, the changes need specific visualization means.

In this paper, we use the modern cases of transformation of ERP (Enterprise Resource Planning) systems to the Best of Breed solutions and an architecture modeling language ArchiMate to propose a framework for visualization of changes. The framework includes a new abstraction called "Gap of Changes", artifacts, principles and means for visualization. The new abstraction is defined on a metamodel that makes it reusable in different enterprise and software architecture description languages. The framework is tested with real cases of changes of ERP using the Best of Breed strategy.

1 Introduction

Modern enterprises are systems of business processes, software applications and technology. The enterprise architecture description languages include software architecture languages and allow for presentation the snapshots of concepts/relations of an enterprise architecture at one specified moment.

Enterprises are often changed or transformed in order to survive and preserve their market positions [8]. The changes of enterprises that should be implemented, are reflected by two enterprise architectures: As-Is and To-Be; or by two sets of architectural views: As-Is and To-Be. Understanding the changes becomes similar to a game of "finding changes in two drawings". As the size of the enterprise architecture and the number of view pairs grows, the changes can be overlooked or ambiguously understood by several implementation teams. In order to support the unambiguous understanding of changes by the implementation teams, the changes need specific visualization means.

B. Shishkov (Ed.): BMSD 2016, LNBIP 275, pp. 140–160, 2017.
DOI: 10.1007/978-3-319-57222-2_7

In this paper, we use the popular enterprise architecture language ArchiMate [10] to explore the difficulties during the visualization process, and the abstractions needed for it. Our choice of the ArchiMate modeling language is explained by several reasons.

- First, ArchiMate follows to the TOGAF (the abbreviation of "The Open Group Architecture Framework"). It is a standard for enterprise architecture [11].
- Second, ArchiMate is based on a metamodel that contains a layered structure in which the business, application and technology architectures are covered and the modeling of relations between the different layers is supported. Having means to model different domains of the same business, ArchiMate fills the gap between the different domain architectures and the missing relationships between these architectures, as they are most of the time created by separate architects in different modeling languages [5].
- Third, the ArchiMate language is used in industry, and it is supported with some open-source tools, for example, Archi 2.4. [4]

We investigate the ArchiMate and its extensions.

As a result, we propose a visualization framework based on the ArchiMate and its extensions. The proposed framework is tested with the cases of transformation of ERP (Enterprise Resource Planning) systems using the Best of Breed strategy to illustrate the visualization of changes.

Our framework contains a new abstraction for visualization of changes, artifacts, principles and means for visualization. We explain the new abstraction on a metamodel extending the metamodel of ArchiMate. Such an approach makes our result applicable in other architecture description languages based on the TOGAF standard.

The structure of the paper is the following.

Section 2 describes some characteristics of ArchiMate and its extensions relevant to the visualization of changes.

Section 3 presents our framework, the definition and the metamodel of a new abstraction called "*Gap of Changes*" for visualization of changes.

Section 4 describes a test case of transformation of an ERP system using the Best of Breed strategy.

Section 5 visualizes the strategic views on the test case.

Section 6 uses our framework to visualize the test case at the application layer.

Section 7 visualizes the test case at the technological layer.

Section 8 discusses the principles, means and abstractions used for visualization and the scalability issues.

Section 9 concludes the paper.

2 ArchiMate and Its Extensions

ArchiMate is one of the popular tools for visualization of enterprise architecture [3,5]. Mostly, it is because it supports visualization of three related layers of the enterprise architecture:

1. the business layer (products, services, actors, processes);
2. the application layer (application components, application functions and data objects) and
3. the technological layer (infrastructure services, hardware, system software).

The visual elements of ArchiMate within the application and technology layers are mostly concepts and relations [6]:

- Active elements performing behavior (i.e. application service, application component).
- Elements describing behavior (i.e. infrastructure service, application function).
- Passive elements on which behavior is performed (i.e. data object).
- Relations between the elements, depicted as connecting lines between elements or boxes.
- The visual elements can be included into other elements by aggregation, composition and grouping relations.

There are also ArchiMate extensions [10] that cover high level concepts used in business implementation. Among them are the motivation extension and the implementation and migration extension. These extensions contain

- Elements describing motivation: goals, requirements, principles;
- Elements describing migration: plateaus and gaps.

The elements used in this paper are listed in Fig. 1.

Our research goal is to define and test a framework for visualization of architecture changes. The notions of plateau and gap in the migration extension should be related to changes.

"A plateau is defined as a relatively stable state of the architecture that exists during a limited period of time" [10]. Plateaus can be used to present the As-Is and To-Be view.

"A gap is defined as an outcome of a gap analysis between two plateaus" [10]. Figure 2 is a fragment of the ArchiMate metamodel. It shows that a gap is only associated with core elements combined in different plateaus. So, it is supposed that an analysis of the differences between the As-Is and To-Be architectures results only a set of changed elements. In this paper, we show that not only changed elements should be captured to visualize architectural changes.

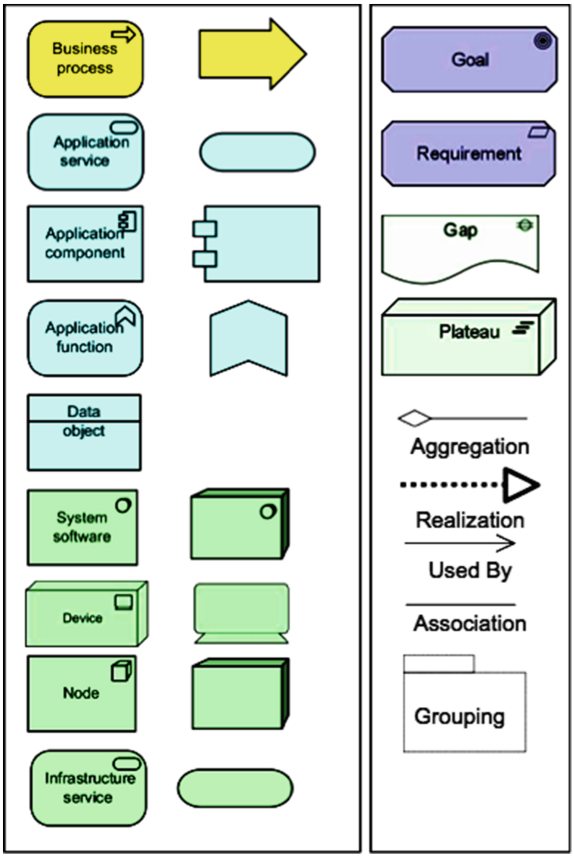

Fig. 1. Visual elements of ArchiMate [10] and its extensions used in this paper

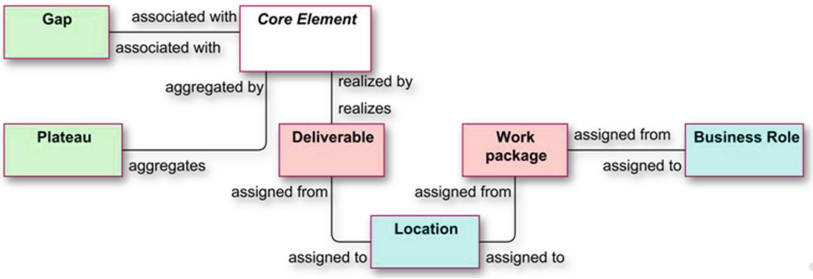

Fig. 2. A fragment of the metamodel for the migration extension: Gap, Plateau and Core Elements [10]

3 Framework for Visualization of Changes

In fact, an analysis of the differences between the As-Is and To-Be architectures results in more than a set of changed elements.

First, we suggest that a gap not only can *be associated with core elements* as it is defined in the migration extension of ArchiMate, but also can *aggregate* a selection of core elements from different plateaus (As-Is and To-Be plateaus).

Second, the analysis of two plateaus may result in several sets of tuples combining old and new architectural elements at all layers from the business layer to implementation layer and combinations of layers. In one set of tuples the new elements will replace the old ones, in other set of tuples, the new elements will modify the old ones. There will be sets of obsolete elements and completely new elements. There are no such tuples in the ArchiMate metamodel.

Third, the relations between the old and new elements are not defined in the set of the ArchiMate relations.

We formalize the result of an analysis of the differences between an As-Is and an To-Be architectures as a new abstraction that we call *Gap of Changes*.

3.1 Abstraction *Gap of Changes*

Each ArchiMate-model can be considered as a graph with nodes and ordered pairs of nodes. A node is depicted as a box or an area. A directed pair of nodes has different forms of presentation from arrows between areas to depicting one node in the area of another (Fig. 1). In the ArchiMate-tools, the nodes are maintained as objects and the ordered pairs are presented as binary relations in a "repository". They are used in different views.

An enterprise architecture is a tuple of objects and relations defined on the set of these objects.

$$Arch = (O, R).$$

In case of a transition, we distinguish two ArchiMate-models: an As-Is model and a To-Be model.

$$AsIs = (O_{AsIs}, R_{AsIs}),$$
$$ToBe = (O_{ToBe}, R_{ToBe}).$$

Some objects and relations exist only in the As-Is model, others appear only in the To-Be model, and, usually, a significant amount of objects and relations exists in both models.

We distinguish:

- An object or a relation is called *obsolete* if it exists in the As-Is model and not in the To-Be model.
- An object or a relation is called *new* if it exists in the To-Be model and not in the As-Is model.
- An object is called *changed* if it exists in both models, and it is linked (by relations) in the To-Be model to another set of objects than in the As-Is model.

– An object is called *unchanged* if it exists in both models, and it is linked (by relations) in the To-Be model to the same set of objects as in the As-Is model.

In Archimate, we designate some views as "As-Is". We make sure that an "As-Is" view contains no new elements. We can also designate views as "To-Be", making sure that an "To-Be" view contains no obsolete elements.

An abstraction called *Gap of Changes*, *Gch*, is a view that

– combines the obsolete, new, changed objects and relations,
– annotates them with new relations <Replaced-By> and <Extended-By> and
– positions the obsolete and changed objects in the As-Is architecture and the new and changed objects in the To-Be architecture by showing a subset of unchanged objects related to the obsolete, new and/or changed objects.

A *Gap of Changes* is a tuple:

$$Gch = (O_{obsolete}, O_{new}, O_{unchanged}, O_{changed}, R_{obsolete}, R_{new}, R_{replaced-by}, R_{extended-by}, R_{border})$$

– $O_{obsolete} = \{o |\ o \in O_{AsIs}\ and\ o \notin O_{ToBe}\}$
is a set of obsolete objects from the As-Is architecture.
– $O_{new} = \{o |\ o \notin O_{AsIs}\ and\ o \in O_{ToBe}\}$
is a set of new objects from the To-Be architecture.
– $O_{unchanged} = \{o |\ o \in O_{AsIs}\ and\ o \in O_{ToBe}\ and\ \forall x : (o, x) \in R_{ToBe} \Leftrightarrow (o, x) \in R_{AsIs}\}$
$O_{unchanged}$ is a subset of all unchanged objects. They have the same relations in the To-Be model in comparison with the As-Is model. These objects are included to relate the *Gap of Changes* to the As-Is and To-Be architectures.
– $O_{changed} = ((O_{AsIs} \cap O_{ToBe}) \setminus O_{unchanged})$
is a set of changed objects that appear in both models and do not have the same relations in both models.

All sets of objects are disjoint.

– $R_{obsolete} = \{(a, b) | (a, b) \in R_{AsIs}\ and\ (a, b) \notin R_{ToBe}\}$
is a set of obsolete relations that appear in the As-Is model and do not exist in the To-Be model. $R_{obsolete}$ relate obsolete and/or changed objects.
– $R_{new} = \{(a, b) | (a, b) \notin R_{AsIs}\ and\ (a, b) \in R_{ToBe}\}$
is a set of new relations defined on the sets of new and changed objects.
– $R_{<replaced-by>} \subseteq O_{obsolete} \times O_{new}$
is a set of annotations, being relations of type $< Replaced - By >$.
These relations are added during the analysis of the difference between the As-Is and To-Be models.

Not all obsolete elements are necessarily replaced by new elements.
Not all new elements replace the obsolete ones.

– $R_{extended-by} \subseteq O_{changed} \times O_{new}$
  is a set of annotations, being relations of type $< Extended - By >$.
  These relations are added by analysis of the difference between the As-Is and To-Be models.
– $R_{border} \subseteq (O_{unchanged} \times O_{changed}) \cup (O_{changed} \times O_{unchanged})$
  is a set of relations between the changed elements and the unchanged elements within the As-Is and To-Be architectures.

Note that $O_{obsolete}$ or O_{new} can be empty.

3.2 Metamodel of a *Gap of Changes*

Figure 3 shows the metamodel of our framework for visualization of changes.

We separate the As-Is Plateau and To-Be Plateau conceptually. Each of those plateaus aggregates own set of core elements: As-Is Core Element and To-Be Core Element. The intersection of those sets is usually not empty.

A *Gap of Changes* aggregates: the <obsolete> and <changed> Core Elements from an As-Is plateau and the <new> and <changed> Core Elements from a To-Be plateau.

We add new types of relation between the Core Elements aggregated by a *Gap of Changes*: <Replaced−By> and <Extended−By>.

Such an approach allows one to select an abstraction representing changes in a *Gap of Changes*, so that the communication teams do not need to find differences in two architectures As-Is and To-Be.

Relations between the As-Is and To-Be Core Elements aggregated in the *Gap of Changes* provide the possibility to associate the constraints, requirements and principles with the gap elements and, in such a way, visualize them.

Separation of two plateaus also allows one to associate goals and drivers of transformation with the To-Be plateau. The concepts of goals and drivers are also taken from the motivation extension of ArchiMate [10].

3.3 Visual Means: Colors and Labels

Our framework suggests to use colors and labels to visually separate the new, changed and the obsolete elements of a gap of changes. We suggest to use colors and show

– new objects in green;
– changed objects in orange;
– obsolete objects in grey.

However, we have found, that when a model contains a large number of objects and even objects over multiple layers, the use of colors can be distracting and can make the visualization unclear, due to the fact that the layers in ArchiMate are distinct by color. So, we suggest to include the labels <*new*>, <*changed*> and <*obsolete*> in the objects name.

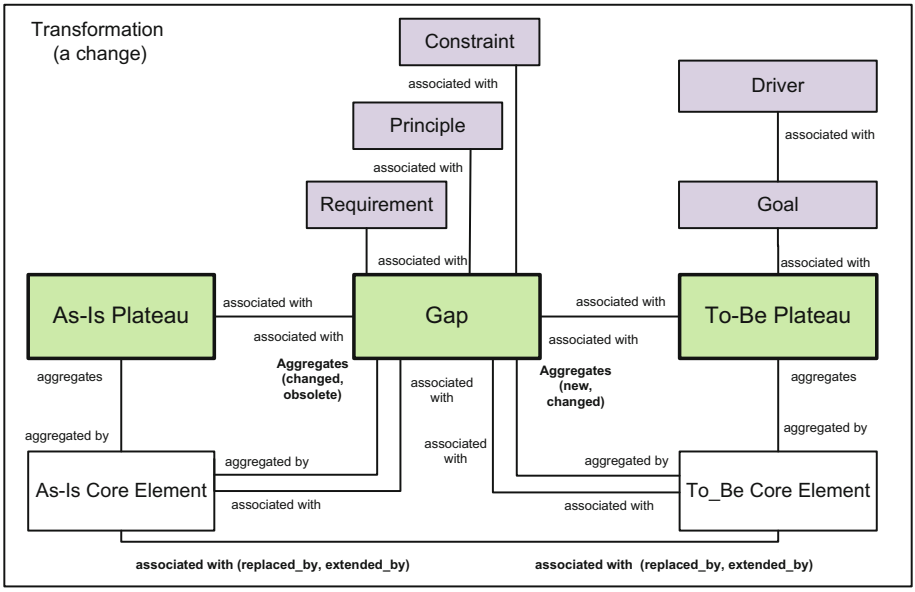

Fig. 3. The metamodel of a framework for visualization of changes

3.4 Border Relations

The abstraction *Gap of Changes* includes the border relations for reminding the place of the gap elements in the As-Is and To-Be architectures.

The relations used for reminding are often fall into the categories of group, aggregation, composition and derived relations. For example, the reminder may show that the elements of a *Gap of Changes* belong to a specific layer or a component. Let us recall that "the grouping relationship is used to group an arbitrary group of model objects, which can be of the same type or of different types. In contrast to the aggregation or composition relationships, there is no "overall" object of which the grouped objects form a part. Unlike the other language concepts, grouping has no formal semantics. It is only used to show graphically that model elements have something in common. Model elements may belong to multiple (overlapping) groups." [10] Aggregation and composition in ArchiMate have the formal semantics inspired by the UML. [9] When elements belong to other elements by an aggregation or composition relation, elements can be *rolled up to the aggregated or composed element.*

Derived relations. It may be convenient to a skip the intermediate relations and elements in a certain chain and show only a very well known element of a component. This can be done using the derived relations of ArchiMate. The structural relations in ArchiMate are divided into four categories of strength, where "association is the weakest structural relationship; composition is the strongest. Part of the language definition is an abstraction rule that states that

two relationships that join at an intermediate element can be combined and replaced by the weaker of the two." [10] Using this abstraction rule of derived relations, a view may abstract from the intermediate elements of a chain of related elements (make some intermediate elements invisible). Elements can be rolled up, using the derived relationship of ArchiMate, where the chain of related elements can be generalized by relating two elements in the chain using the "weakest" relation in the chain.

3.5 Visual Artifacts

On the basis of the metamodel of the new abstraction, our framework selects a set of visual artifacts needed for visualization of changes.

All visual artifacts of our framework fall into three categories:

1. Strategies views,
2. As-Is views, To-Be views and
3. Views on a *Gap of Changes*.

Any change depends on the chosen strategy and on the given requirements. Therefore, we need a goal/requirements view associated with a change presented as one transition from an abstract As-Is to an abstract To-Be.

The As-Is views, To-Be views and views on a *Gap of Changes* may be visualized at different system layers: the business process layer, at the application layer and at the technology layer. The elements of the views on a *Gap of Changes* at different layers may be related in order to form the visualization at a combination of layers. For example, it is good to remind that the changes are implemented for business. Therefore, the relations between a *Gap of Changes* at the application and implementation layers are related to the business layer.

4 Testing Case: Transformation of ERP Using the Best of Breed Strategy

In order to test the proposed framework, we set an experiment for visualization of the ERP implementation within Royal Vopak (https://www.vopak.com/) modified using the Best of Breed strategy [7].

4.1 Best of Breed

Business processes often contain some parts of functionality that are well supported by standard solutions. The Best of Breed strategy is directed to use the most suitable standard software and to develop only the parts that are not supported by standard software [7]. This approach promises flexible application implementation, low costs of maintenance and changes [2].

Many companies have ERP systems that provide all the applications for an enterprise and integrate them in a superior solution where every module may

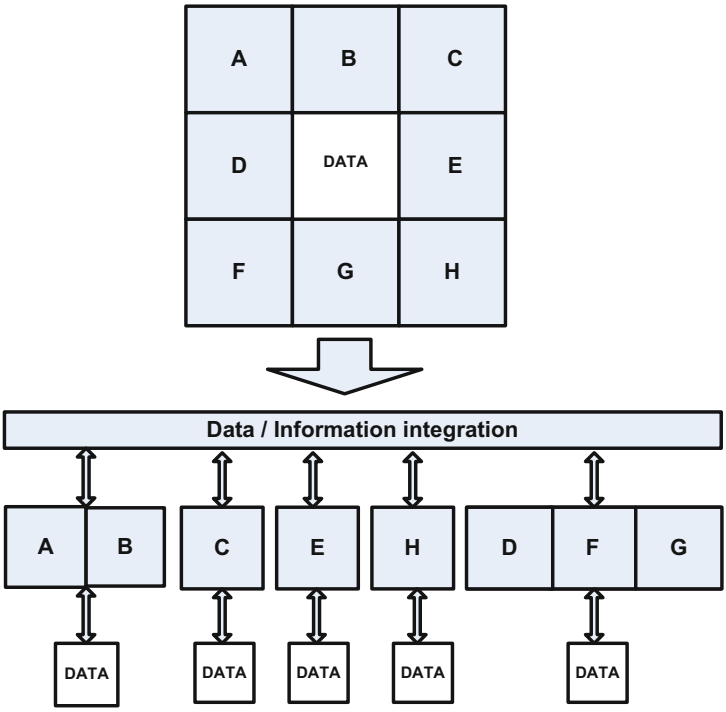

Fig. 4. From ERP to Best of Breed: The ERP implementation of functions (A, B, C, D, E, F, G, H) are decomposed into the best applications available on the market (A, B), (D, F, G), C, E and H.

not be the best of its class. If a company has an implemented ERP system and wants to change it in order to use the best standard solutions, the company should examine and change its architecture.

Ideally, the transformation is the decomposition of the application functionality and the data, as it is shown in Fig. 4. In reality, the elements are not only decomposed, but also removed, added and changed. Because of that, we have studied several cases of architectural changes.

We consider the following business situations that cover the transformation from ERP to Best of Breed:

1. The business process remains unchanged, but the changes should be made in the application and the technological layers by adding, changing and removing elements.
2. The relations between applications may change.
3. The relations between technological elements and applications may change.

For each of the situations, we analyse what should be visualised. Doing the experiment, we have discovered the repeated steps, artifacts, principles and means use-

ful for visualization of changes e and also two missing relations <Replaced−By> and <Extended−By> that we included into our Framework.

4.2 Replacement of an Order Management Service

In this paper, we present the results of visualization of the testing case of replacement of an Order Management Service.

As-Is situation. Let a company have an Order Management Service allowing one to take orders from clients face-to-face and by phone.

To-Be situation. The Order Management Service should be replaced with the new Order Management Component, so that the orders can be taken via e-mail and B2B channels.[1]

The goal of the transformation of ERP using the Best of Breed strategy is to replace the generic functionality that is bound in an ERP system by separate, domain specific, (cloud) applications that can support a specified business process. It is preferable that all new components can be found as the best components in terms of the user interface and performance.

5 Strategic Views

The strategic views usually assign a name to a transformation and formulate the goals of transformation.

Figure 5 visualises the "Replacement of an Order Management Service" using plateaus and gap elements from the migration extension of ArchMate.

Fig. 5. Strategic view on the "Replacement of an Order Management Service"

Figure 6 visually presents the requirements for the transformation.

The set of strategic views may be extended in the correspondence with the well accepted goal-oriented approaches [1]. The goal views and views on requirements may be related with the gap or with the To-Be architecture. The visual means for the strategic views on the transformation are available in the migration extension of ArchiMate.

[1] Business-to-business (B2B) means that the services or goods are sold to other businesses, not to private customers. B2B channels provide wider negotiation opportunities in comparison with the Business-to-Customer channels.

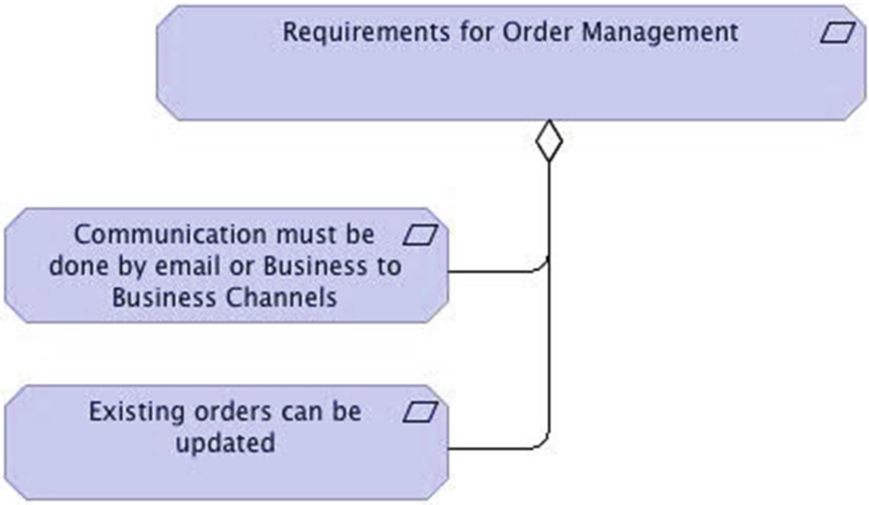

Fig. 6. Requirements for the "Replacement of an Order Management Service"

6 Visualization at the Application Layer

A view of the As-Is architecture may be given before any transformation. In practice, it is often not the case, so in a transformation project both views should be found or depicted.

The As-Is and To-Be architectures can be visualized at selected layer.

In Figs. 7 and 8 the two architectures of the Order Management Service are visualized on the application layer.

For the sake of simplicity, we select only the application layer. All figures are made using the tool Archi [4].

The As-Is architecture is shown in Fig. 7.

The To-Be architecture is depicted at Fig. 8.

The abstraction *Gap of Changes* is shown in Fig. 9. The new architectural elements are shown in green, the obsolete elements are grey.

The `External Communication` is an obsolete module. It is <Replaced-By> the new elements `E-mail Communication` and `B2B Communication`.

The `Order Entry Module` belongs to both the As-Is and To-Be architectures. It is <Extended-By> the new element `Order Updating`.

In order to position the *Gap of Changes* in the As-Is architecture, the component `Order Entry Module` is used. The component `Order Entry Module` groups the obsolete element `External Communication` and the new element `Order Updating`.

In order to relate the *Gap of Changes* with the To-Be architecture, the component `Communication Application` is included into the abstraction to group

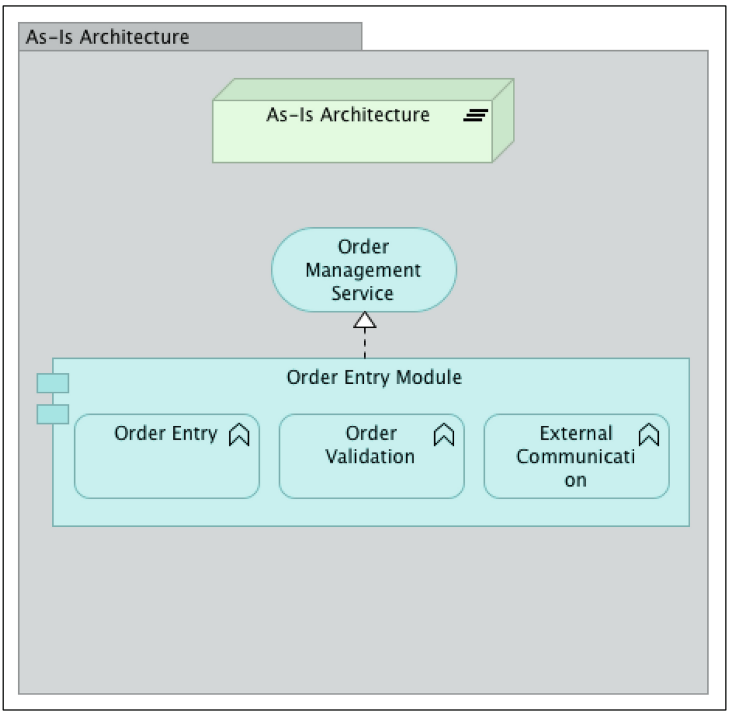

Fig. 7. As-Is Architecture of the "Replacement of an Order Management Service"

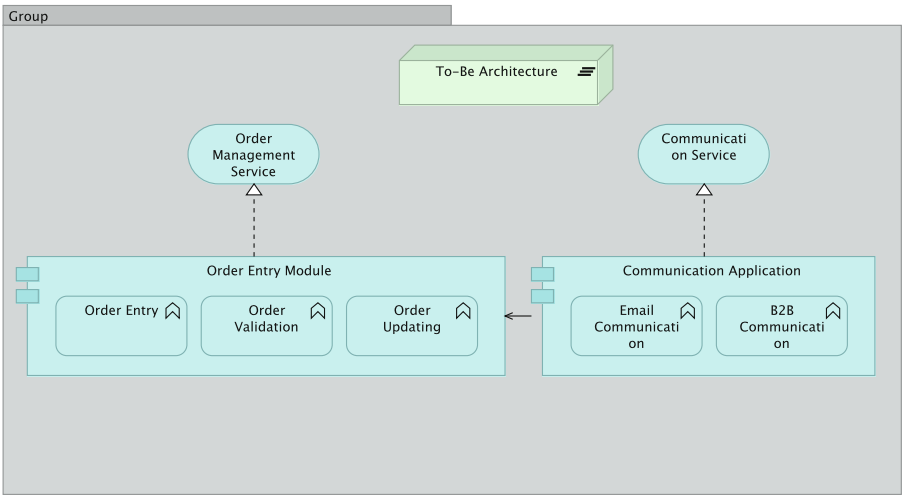

Fig. 8. To-Be Architecture of the "Replacement of an Order Management Service"

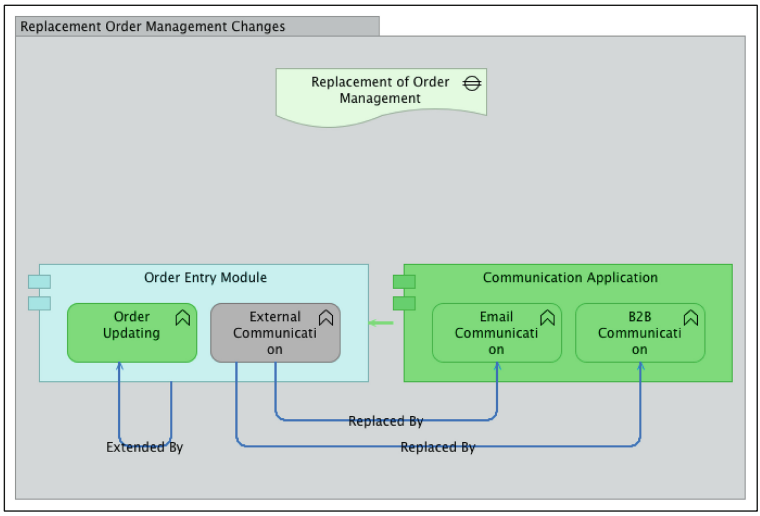

Fig. 9. Gap of changes with focus on relations between the old and new application functions (Color figure online)

`E-mail Communication` and `B2B Communication`. In order to minimize the number of elements in the `Gap of Changes`, the software architects have not shown the `Communication Service` and the `Order Management Service`.

The abstraction *Gap of Changes* is defined by the following tuple of sets:

- $O_{obsolete}$ = {External Communication}.
- O_{new} = {E-mail Communication,
  B2B Communication,
  Order Updating,
  Communication Application}.
- $O_{changed}$ = {Order Entry Module}.
- $R_{obsolete}$ = {(Order Entry Module, External Communication)},
- R_{new} = {(Communication Application, Order Entry Module),
  (Communication Application, E-mail Communication),
  (Communication Application, B2B Communication),
  (Order Entry Module, Order Updating)}.
- $R_{<replaced-by>}$ =
  {(External Communication, E-mail Communication),
  (External Communication, B2B Communication)}.
- $R_{<extended-by>}$ =
  {(Order Entry Module, Order Updating)}.

The sets of unchanged objects and border relations are empty in this example.

7 Visualization at the Technological Layer

7.1 As-Is Architecture at the Technological Level

Figure 10 shows the As-Is architecture on two layers: the Application Layer and the Technological Layer. Such a visualisation allows us to show that our service `Order Entry Module` is bound to the ERP implementation. Namely, the `PERI infrastructure service` and `PERI database`.

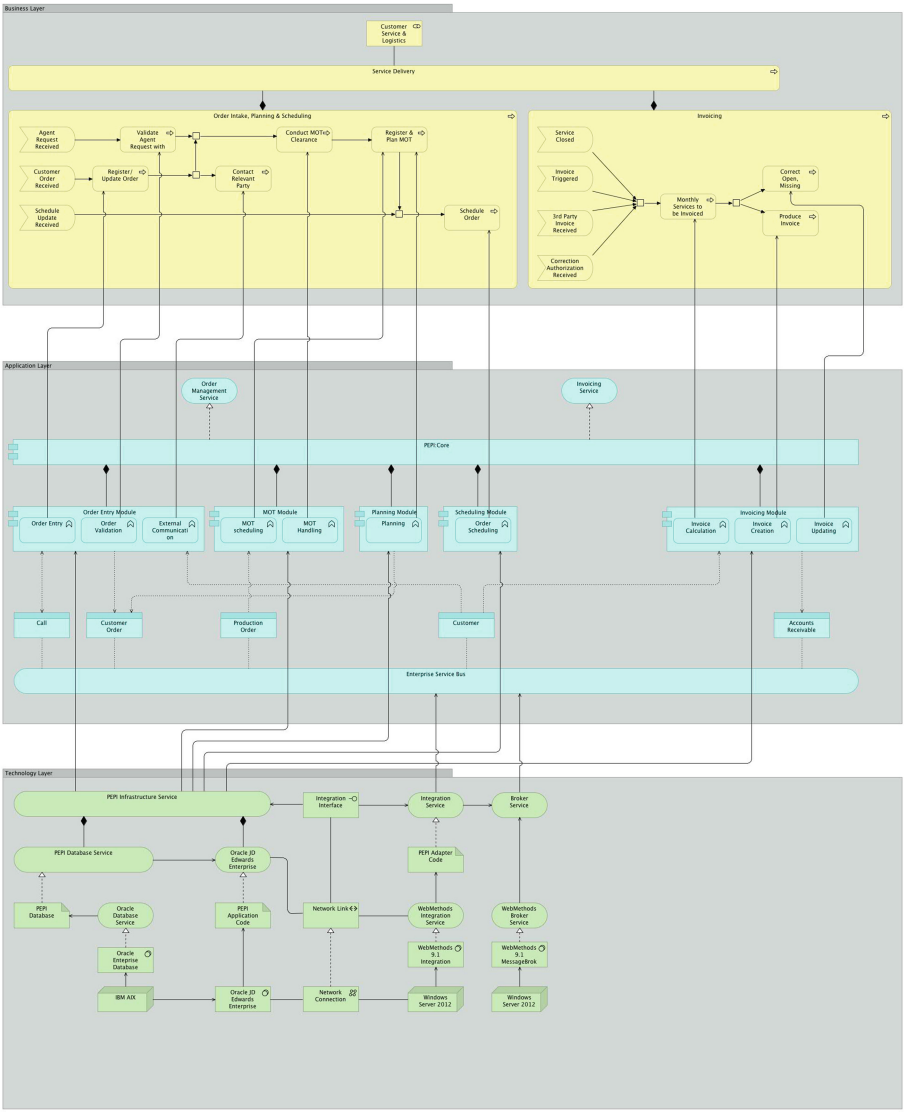

Fig. 10. As-Is at the technological level

7.2 To-Be Architecture at the Technological Layer

The technological goal of the transformation of ERP to Best of Breed is to support the new applications, replacing the generic functionality that is bound in an ERP system, with the cloud access. Figure 11 shows that all services should become the cloud applications.

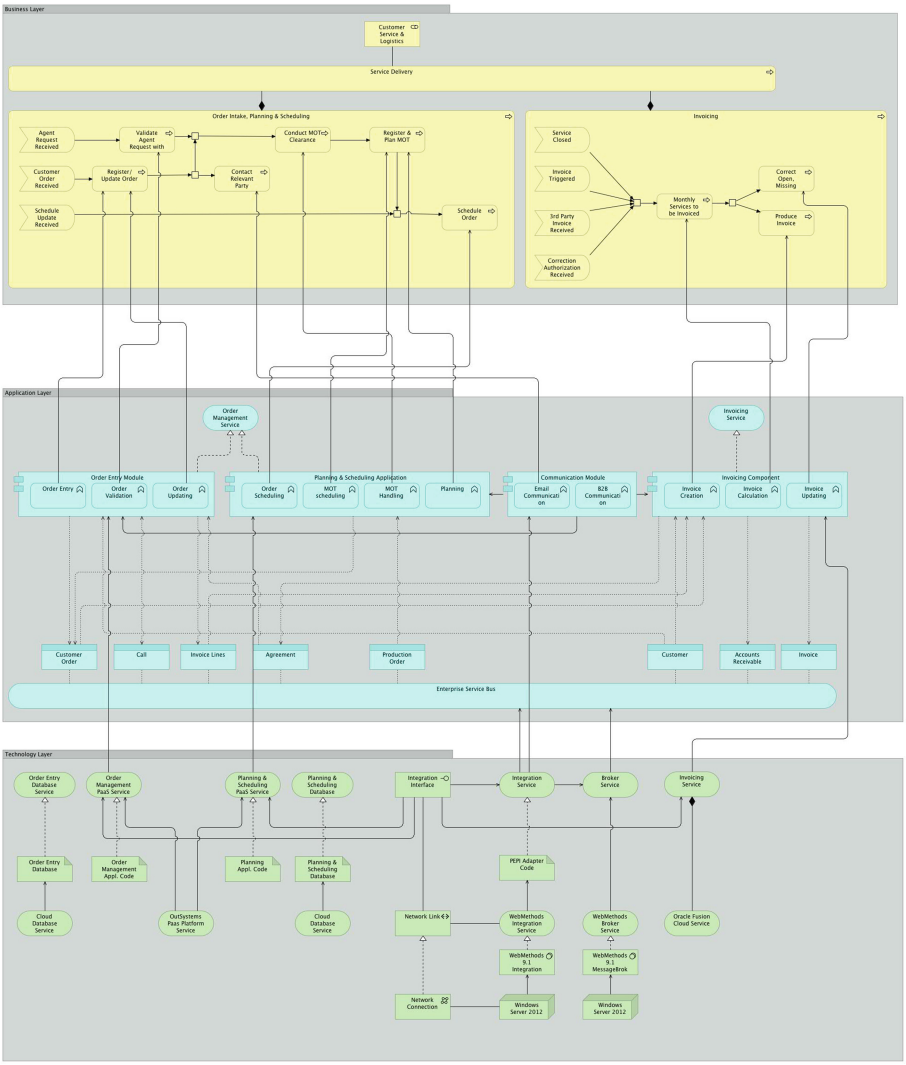

Fig. 11. To-Be at the technological level

7.3 Gap of Changes at the Technological Layer

Summarising the observation of the As-Is and To-Be architectures, we result in the *Gap of Changes* shown in Fig. 12.

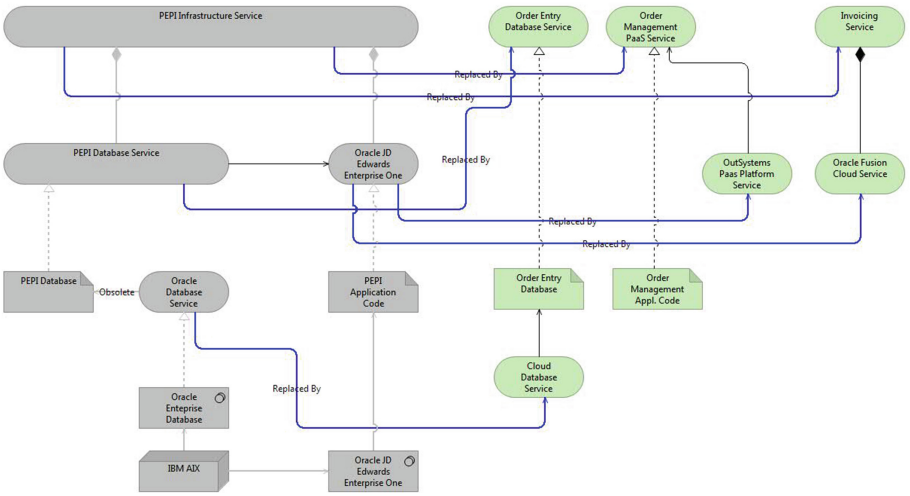

Fig. 12. Gap of changes at the technology level (Color figure online)

The abstraction *Gap of Changes* at the technological layer is described as follows:

- $O_{obsolete}$ = {PERI Infrastructure Service,
  PERI Database Service,
  PERI Database,
  Oracle Databases Service,
  Oracle Enterprise Database,
  IBM AIX,
  Oracle JD Edwards Enterprise One,
  PERI Application Code,
  Oracle JD Enterprise One (Data)}.
- O_{new} = {Order Entry Database Service,
  Order Management PaaS Service,
  Invoicing Service,
  Oracle Fusion Cloud Service,
  Out System Paas Platform Service,
  Order Management Appl. Code,
  Order Entry Database,
  Cloud Database Service}.
- $R_{<obsolete>}$ - all relations between obsolete (grey) objects in Fig. 12.
- $R_{<new>}$ - all relations between new (green) objects Fig. 12.

– $R_{<replaced-by>} = \{$
  - (PERI Infrastructure Service, Invoicing Service),
  - (PERI Infrastructure Service, Order Management PaaS Service),
  - (PERI Database Service, Order Entry Database Service),
  - (Oracle JD Edwards Enterprise One, Out System Paas Platform Service),
  - (Oracle JD Edwards Enterprise One, Oracle Fusion Cloud Service),
  - (Oracle Databases Service, Cloud Database Service)}.
– Other sets are empty in this case.

In the case of automation of visualization of a *Gap of Changes*, the type of each relation can be added to each relation as a description attribute.

8 Discussion

8.1 Visualization Principles

Our visualization principles are the minimum visual elements and the focus on changes. In order to follow these principles, we sought for means for separation the changes and unchanged elements and for visualization of relations between unchanged elements and new elements at the business, application and technology layers. We explored available means of visualization in ArchiMate.

8.2 Omitting Unchanged Elements

The unchanged elements can be sometimes omitted from the model. When a model contains unchanged elements and those elements are part of a more generic element (that is required for clarity in the model), the elements can be left out and only the generic element can be a part of the model, using the generalization method.

In Fig. 12 we omit the element representing the planning functionality `Planning and scheduling application`. This application and its implementation is not relevant for the applications `Order Entry Module` and `Communication Module` and may be omitted.

8.3 New Relations Between Changed Elements and Motivation

We have found it very useful that the core elements of ArchiMate can be related to motivational elements (goals, principles, requirements etc.) [10] For example, the principle, ‘‘`Cloud unless`’’ can be chosen to motivate the prosed technological elements `Oracle Fusion Cloud Service` and `Cloud Database Service` (Fig. 12).

8.4 Relations <Extended-By>, <Replaced-by>

The core of the transformation of ERP using the Best of Breed strategy is the decomposition of functionality and data, as it is shown in Fig. 4. However, in reality, this pure transformation is rare. Usually, even when following the Best of Breed strategy of changes, some functions are deleted; the new functions replace the old ones, and the new functions are often added. Therefore, we have found it useful to introduce the relations between the new and old functions and groups of functions.

Relations like <Extended-by> and <Replaced-by> are definitely needed for the visualization of changes. As the current ArchiMate language and the corresponding tools do not have such relation types, we reuse the directed relations of types <used-by> or <triggering relationship> and replace their labels by <Extended-by>, <Replaced-by>.

8.5 Scalability of Visualization

In all cases, the visualization of changes is a very creative activity that demands good abstraction and generalization skills. The comprehensible views of a system with large amount of elements cannot be produced without abstractions. In particular, the ability to make groups of elements and the ability to responsibly omit elements are the key techniques for visualizing of changes in a comprehensible way.

Three types of abstractions have been found useful for visualization of changes:

1. the abstractions from the unchanged elements;
2. the abstraction from the elements that are out of focus of particular view;
3. the abstractions from relations and elements in a chain using the derived relationships of ArchiMate.

For example, the derived relationship rule allows us to show in Fig. 13 only the access relations between the services the *Order Entry Module* and the *Communication Application* and the messages (data objects) sent via the *Enterprise Service Bus*. All other technological components are omitted. Indeed, at the border of the application and technological layers, there is the Enterprise Service Bus. The new and modified services suppose to subscribe for and publish messages using the Enterprise Service Bus. Therefore, we present a *Gap of Changes* view with the focus on communication elements. Figure 13 shows that the `Communication service` are subscribed for the `Customer Orders` and `Invoice Lines` published by the `Order Management Service`.

Another way of making the visualization of changes scalable is showing a sequence of gaps. As changes within a system are often implemented in steps, the visualization of each step as a *Gap of Changes* may restrict the number of changed elements and relations and make the visualization of each gap comprehensible.

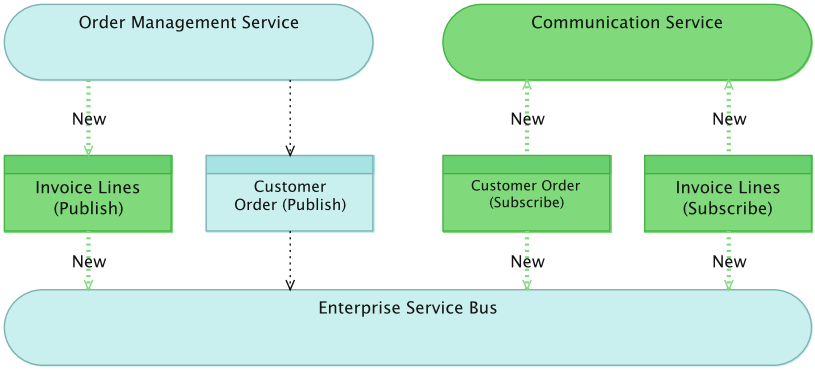

Fig. 13. Gap of changes with focus on communication elements

9 Conclusions and Future Work

This paper presents a framework for visualization of changes in ArchiMate. Our framework defines an abstraction of two architectures, As-Is and To-Be. This abstraction is called *Gap of Changes*. The new abstraction uses two architectures and two new relations to express a replacement or an extension of one element (or a group) from As-Is architecture by an element (or a group) in the To-Be architecture. The advantage of the proposed abstraction is the separation of the analysis of changes in two architectures as a directed business activity that results in a view on a *Gap of Changes*. This view can be unambiguously understood by implementation teams and can be used for planning and management. The proposed abstraction has been precisely defined to be built into the tools for visualization of enterprise architectures.

The framework has been tested with cases of transformation of ERP using the Best of Breed strategy. We expect that different combination of layers may be used for visualization of changes driven by different strategies. However, the core of the proposed framework, the abstraction *Gap of Changes* will remain the same. In the future work, we are going to apply this framework in new projects of changes with different strategies. We also plan to use the abstraction *Gap of Changes* to capture the patterns of changes corresponding to different strategies.

References

1. van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, Chichester (2013)
2. Cardoso, J., Bostrom, R.P., Sheth, A., Sheth, C.I.A.: Workflow management systems and ERP systems: differences, commonalities, and applications. Inf. Technol. Manage. **5**, 319–338 (2004)

3. Fritscher, B., Pigneur, Y.: Business IT alignment from business model to enterprise architecture. In: Salinesi, C., Pastor, O. (eds.) Advanced Information Systems Engineering Workshops. Lecture Notes in Business Information Processing, vol. 83, pp. 4–15. Springer, Heidelberg (2011)
4. Institute of Educational Cybernetics, Archi 2.4 (2012). http://archi.cetis.ac.uk/
5. Lankhorst, M.M., Proper, H.A., Jonkers, H.: The architecture of the archimate language. In: Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Ukor, R. (eds.) Enterprise, Business-Process and Information Systems Modeling. Lecture Notes in Business Information Processing, vol. 29, pp. 367–380. Springer, Heidelberg (2009)
6. Lankhorst, M.M., Proper, H.A., Jonkers, H.: The anatomy of the ArchiMate language. IJISMD **1**(1), 1–32 (2010)
7. Light, B., Holland, C.P., Wills, K.: ERP and best of breed: a comparative analysis. Bus. Process Manag. J. **7**(3), 216–224 (2001)
8. Mckeown, I., Philip, G.: Business transformation, information technology and competitive strategies: learning to fly. Int. J. Inf. Manag. **23**(1), 3–24 (2003)
9. OMG, Unified Modeling Language: Superstructure version 2.1.1 formal/2007-02-03 (2003)
10. The Open Group. ArchiMate 2.1 Specification (2013). http://pubs.opengroup.org/architecture/archimate2-doc/chap03.html
11. The Open Group. TOGAF, The Open Group Architecture Framework, Version 9.1, an Open Group Standard (2016). http://pubs.opengroup.org/architecture/togaf9-doc/arch/