Enterprise Modelling and Information Systems Architectures
Vol. 13, No. 1 (2018). DOI:10.18417/emisa.13.1

Special Issue Wirtschaftsinformatik 2017

Enterprise Modeling in Support of SOA Migration Analysis

Sybren de Kinderen[*,a], Monika Kaczmarek-Heß[a]

[a] University of Duisburg-Essen, Universitätsstrasse 9, D-45141, Essen, Germany

Abstract. *This paper shows how enterprise modeling can support a Service-Oriented Architecture (SOA) migration analysis in terms of (1) IT infrastructure understanding, (2) identifying and refining candidate services by means of analyzing both the current IT infrastructure capabilities and business concerns, as well as (3) understanding how candidate services build on the current IT infrastructure. Based on requirements derived from a conducted literature study on SOA analysis and SOA migration projects, we identify the Multi-Perspective Enterprise Modeling (MEMO) method as a suitable language family to support a SOA migration analysis. Furthermore, we extend MEMO's language for IT infrastructure modeling, called ITML, with concepts central to SOA migration, and show how the modeling language can support key phases of a SOA migration project. We also provide a threefold evaluation of our SOA migration modeling approach by means of (1) application to documented SOA migration projects, (2) a scenario-based comparison with ArchiMate, another language that is a promising candidate for a SOA migration analysis, and (3) an assessment against the identified requirements. Finally, we discuss corresponding software tool support.*

Keywords. Enterprise modeling • Service-orientation • SOA migration • Multi-Perspective Enterprise Modeling

Communicated by A. Koschmider. Received 2017-03-30. Accepted after 2 revisions on 2017-10-16.

1 Introduction

Service-Oriented Architecture (SOA) is positioned as an instrument to foster organizational flexibility and agility (Alwadain et al. 2016; MacLennan and Van Belle 2014), e. g., by reducing time-to-market, reusing assets across different lines of business, and shortening project lead times (Rosen et al. 2008). As implied by its name, a service is the primary object of interest for a SOA. A service is defined as a "self-contained" module that provides a business functionality via a standardized interface, whereby the interface hides how the functionality provided by a service is realized (Papazoglou et al. 2008).

The self-contained nature of services offers several advantages. For one, service-orientation promotes loose coupling (Rosen et al. 2008, p. 64), which implies that a change in one software component has a minimal effect on the remaining software components. As a concrete example, Rosen et al. (2008, p. 6) discuss how service-orientation adopted by a bank allowed for reuse of IT functionality, which in turn accelerated the offering of a banking service across different organizational channels, be it internet banking, mobile banking, or others.

Importantly, service-orientation should help bridge the gap between business concerns and IT concerns (Bhallamudi and Tilley 2011; Papazoglou and Heuvel 2006). The mentioned gap results, among others, from the technology-driven functional silos of previously designed IT systems, i. e., legacy systems, which are often developed over decades using such programming languages as COBOL, C, or C++ (Sneed et al. 2016). Although legacy systems are usually hard to modify and expensive to maintain, at the same time, they

* Corresponding author.
E-mail. sybren.dekinderen@uni-due.de
Note: This work is based on de Kinderen and Kaczmarek-Heß (2017).

are critical for the mission of the organization and thus, must be operational at all times (Khadka et al. 2013b; Sneed et al. 2016).

Migration of legacy systems to service-based systems enables achieving advantages offered by SOA, while, at the same time, reusing the required capabilities of the legacy systems (Razavian and Lago 2010). Roughly speaking, a SOA migration project, i. e., a move towards service-orientation, encompasses two main steps: (1) determining which elements of an IT infrastructure should be migrated, and (2) deciding on how the migration itself should be performed (cf. e. g., Khadka et al. 2013b; Razavian and Lago 2010).

However, successfully migrating an organization towards service-orientation has proven to be challenging (Bhallamudi and Tilley 2011; Hirschheim et al. 2010; Khadka et al. 2014; Rabelo et al. 2015). Among others, this is due to insufficiently addressing: (1) *an alignment of business and IT concerns* related to, e. g., an insufficient understanding of how SOA should support business concerns, such as, e. g., business processes to be supported by service-orientation (Bhallamudi and Tilley 2011; Khadka et al. 2014; Rosen et al. 2008, p. 137); and (2) *technical concerns*, related to, e. g., inadequate integration with legacy systems (cf. Bhallamudi and Tilley 2011; Khadka et al. 2013b; Lewis et al. 2006; Razavian and Lago 2010). Therefore, an instrument is needed that would support analysis of such business and technical aspects in tandem.

In this context, a promising instrument seems to be the application of enterprise modeling (EM), which focuses on supporting sense-making of organizational concerns in tandem with information system concerns (Frank 2012). Enterprise modeling indeed has the potential to play an important role in supporting SOA migration (Winter and Ziemann 2007; Ziemann et al. 2006). In particular, enterprise modeling can support understanding of current IT infrastructure, which provides a baseline of IT capabilities that can potentially be offered as services (Khadka et al. 2013b; Razavian and Lago 2015). Furthermore, EM may help identifying and refining candidate services, based upon aforementioned IT functionalities, as well as an analysis of business concerns. As an example of the latter, one can translate SOA migration objectives such as "share capabilities across different lines of business" into concrete SOA designs. Finally, EM may foster understanding of how candidate services build on the current IT infrastructure (Khadka et al. 2013b; Razavian and Lago 2015).

Nevertheless, although various (enterprise) modeling approaches exist that allow for modeling IT infrastructures and/or expressing a service-orientation from various angles (OMG 2012; Terlouw and Albani 2013; The Open Group 2013, to name a few), these approaches often on purpose forgo the level of detail that is required to analyse IT infrastructure for the needs of SOA migration. In addition, as we detail in Sect. 3, often they insufficiently relate IT concerns and organizational concerns.

Also other existing initiatives fall short when it comes to supporting SOA migration projects. For instance, in tandem with the emerging use of so called microservice architectures, DevOps is often discussed as an approach to help manage a migration to a service-orientation (see, e. g., Zimmermann 2017). Following Lwakatare et al. (2015), we define DevOps as "a set of engineering process capabilities supported by certain cultural and technological enablers. Capabilities define processes that an organization should be able to carry out, while the enablers allow a fluent, flexible, and efficient way of working" (p. 170). Thus, it encompasses a wide set of potential tools and capabilities. However, due to its wide scope DevOps practices are often heterogeneous, fragmented, and difficult to integrate (Wettinger et al. 2014). Also, since DevOps is such a broad term, there are no clear criteria to determine what falls under its umbrella. As a result, although we consider DevOps as a potentially complementary approach, with its orientation towards management, it does not provide the model-based support that we are after.

As a response to the observed research gap, we focus on the following research question: *What*

should be the scope and characteristics of a modeling language/modeling approach able to support a SOA migration analysis? To address this question, in our earlier work (cf. de Kinderen and Kaczmarek-Heß 2017), we made a first step by extending a selected IT infrastructure modeling language called ITML (Heise 2013), being part of the Multi-Perspective Enterprise Modeling (MEMO) language family (Frank 2012). This paper continues this effort by (1) extending the ITML further and emphasizing how the *extended ITML* can be complemented by other languages of the MEMO language family, so as to link the IT perspective to organizational concerns. In particular, we discuss a relationship to business processes (as modeled in OrgML, Frank 2014), and show how goal modeling (by means of GoalML, Overbeek et al. 2015) can help translate SOA migration objectives into concrete SOA designs; (2) showing, based on a set of key SOA migration activities distilled from literature, which role modeling can play across the entire life-cycle of SOA migration analysis. This is opposed to focusing only on the to-be analysis, as in our earlier work (cf. de Kinderen and Kaczmarek-Heß 2017); (3) highlighting the added value of choosing MEMO for supporting the SOA migration analysis compared to other candidate modeling languages. In particular, we do this by a scenario-based comparison of the MEMO language family with the enterprise architecture language ArchiMate. To this end we rely on a realistic, extensively documented SOA migration scenario from the insurance industry, reported by Rosen et al. (2008, pp. 541–578). Finally, (4) discussing software tool support for the extended ITML, and other MEMO languages.

This contribution follows the design-oriented research paradigm (Österle et al. 2010). The resulting IT artifact (i. e., the EM approach MEMO with an extended IT modeling language) aims at providing a benefit to organizations by supporting SOA migration projects. To extend ITML, we follow the language development method proposed by Frank (2010), which has already been successfully used in other projects (e. g., Goldstein and Frank 2016; Overbeek et al. 2015). This method provides a macro-process and corresponding roles, as well as guidelines that support a language designer in the language design process.

This paper is structured as follows. First, we discuss specific features of SOA migration approaches and derive requirements that a modeling language/modeling approach should fulfill. Next, we check the fulfillment of those requirements by existing approaches. Then, we present the proposed extensions to the modeling approach that comes closest to fulfilling our requirements: ITML, being part of the MEMO language family. Subsequently, we evaluate the proposed approach (a) against the defined requirements, (b) by means of an application to an extensively documented SOA migration scenario, and (c) by means of a comparative discussion, in the sense of comparative reflections on modeling the same SOA migration scenario in ArchiMate. The paper concludes with final remarks and an outlook on future research.

2 Modeling Support for SOA Migration Analysis

In the context of the field of Information Systems (IS), "migration" is usually understood as a modernization approach that moves an existing, operational system to a new technological or computing platform, while retaining the data and functionalities of the moved system and causing as little disruption as possible to the existing operational and business environment (cf. Bisbal et al. 1999). Following Almonaies et al. (2010), depending on the required level of system understanding, migration strategies can be classified into two different categories: "black-box" migration (mostly: wrapping) and "white-box" migration (e. g., application re-engineering)[1] . With a black-

[1] We follow a broad interpretation of the term "migration" in line with Khadka et al. (2013a) and Razavian and Lago (2015), encompassing a wide variety of SOA migration strategies, such as wrapping, application re-engineering, and more. This is opposed to Almonaies et al. (2010) who interpret migration as one type of SOA "modernization" strategy.

box strategy, one focuses on the external behavior of a legacy system. Thus, one analyses its inputs and outputs so as to understand the system interfaces, and abstracts away from the inner workings of the system. Differently, as implied by name, white-box strategies require understanding of the legacy system internals. Thus, one requires extensive knowledge about the programming languages used, whether a legacy system is proprietary, etc.

In this section, we review the existing SOA migration approaches. This allows us to discuss the role that modeling can play, and to identify the information required by a SOA migration project. Following this, we derive a set of requirements (cf. Tab. 1) for a language supporting a SOA migration project.

2.1 SOA Migration Analysis and the Role of Modeling

The subsequent analysis is based on a literature review comprising the fields of legacy systems reengineering, SOA migration and development as well as cloud migration. In selecting our sources, we first of all rely on well-established approaches such as, e. g., SOMA (Arsanjani et al. 2008), and SoDD (Papazoglou and Heuvel 2006). Second, we rely on recent literature reviews on SOA migration and cloud migration, such as the work of Jamshidi et al. (2013), Khadka et al. (2013a), and Razavian and Lago (2015); using their aggregated insights, and treating them as "portal" papers for follow-up. For instance, Razavian and Lago (2015) in their systematic literature review identify commonalities and differences between 75 identified approaches and propose a reference model of typical activities that are carried out for the legacy to SOA migration. In turn, Khadka et al. (2013a) provide a historical review of legacy to SOA migration approaches. Based on the evaluation of 121 primary studies the authors conclude that there is a lack of (a) adequate automation level and techniques for determining the decomposability of legacy applications, and, (b) investigations of an organisational perspective on migration. Also, they report on after-migration experience. Finally, Jamshidi et al. (2013) focuses on different approaches to SOA/cloud migration and evaluate 21 studies on cloud migration against different dimensions such as contribution type, evaluation method, means of migration, migration type, migration tasks, intents of the migration, migration tool support, and constraints. In addition to the literature review, we investigate various reported SOA migration projects, e. g., Brandner et al. (2004), Rosen et al. (2008), and Zimmermann et al. (2004).

SOA migration approaches such as Service Migration and Reuse (SMART, Balasubramaniam et al. 2008; Lewis et al. 2005), Service-Oriented Modeling and Analysis (SOMA, Arsanjani et al. 2008), and Service-oriented Design and Development (SoDD, Papazoglou and Heuvel 2006), typically consider a SOA analysis phase to complement SOA implementation and management. Although the emphasis on a SOA analysis differs across the approaches, the idea is similar: SOA migration is as much an organizational as a technical undertaking (Razavian and Lago 2015; Rosen et al. 2008). As such, to complement technical SOA implementation issues (such as a service interface specification, e. g., in terms of Web Service Description Language (WSDL) and various complementary WS-* specifications such as WS-Coordination and WS-Security, cf. Gustavo et al. 2004, p. 140) a SOA analysis supports: (1) aligning services with the business context, in terms of fulfilling organizational goals and/or having a clear role in business processes, and (2) making sure that service-orientation builds on the capabilities provided by existing assets.

Fig. 1 shows the key SOA analysis activities: "Business Context Analysis", "Legacy System Understanding", "Candidate Service Identification and Refinement", and "Target System Understanding". Per activity, we also show the information required, and thus, a role that modeling can play. Those SOA analysis activities have been identified as a synthesis and summary of (a) methods dedicated to SOA migration analysis: SMART – Migration Planning (SMART-MP, Balasubramaniam et al. 2008), a part from the SMART

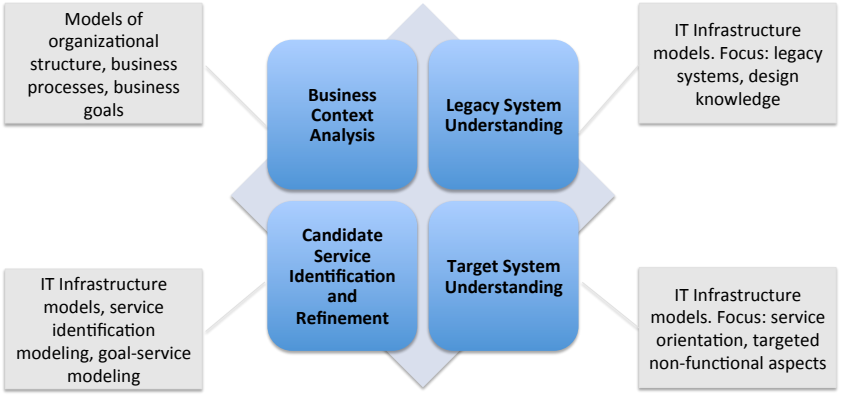

Figure 1: SOA analysis activities – a synthesis and summary based on Arsanjani et al. (2008), Balasubramaniam et al. (2008), Khadka et al. (2013b), and Papazoglou and Heuvel (2006)

SOA migration method family focused on migration planning, and the SOA migration method proposed by Khadka et al. (2013b); as well as (b) the initial phases of the comprehensive SOA migration approaches, such as SOMA, SoDD, and the approach by Rosen et al. (2008). The identified, non-sequential activities are discussed subsequently.

Business Context Analysis. In the initial stage of SOA migration, this activity aims at providing understanding of (1) expectations towards the migration project (Balasubramaniam et al. 2008), and (2) business processes to be supported. In addition, this activity should also already allow to roughly identify candidate services. At a later stage of SOA migration, the "Business Context Analysis" activity is used to support analysis of business concerns within the other activities, such as "Target System Understanding".

In order to understand the business context of SOA migration projects, different aspects of the organization at hand are considered. First and foremost, the SOA migration objectives need to be clarified, and the owners of these objectives should be identified. As we illustrate in Sect. 5.2, this allows to identify conflicts and complementarities between different SOA migration objectives, as well as to translate high-level SOA objectives into more concrete ones. Also the impacted business processes and their role for an enterprise need to be considered. Therefore, the role of modeling is here to provide the required understanding of the above mentioned aspects and facilitate the communication between different stakeholders involved.

Indeed, within this stage of SOA analysis various modeling approaches and languages already have proven to be useful. For example, Rosen et al. (2008) propose to use the Business Motivation Model (BMM, OMG 2015a), which may be helpful to describe the motivations behind the SOA migration project and the stakeholders that have these motivations. However, as BMM has a limited expressive power for modeling motivations, it lacks, e. g., the capability to show conflicts and complementarities.

Furthermore, Papazoglou and Heuvel (2006) suggest to use business process modeling for both eliciting business functionality that is to be supported by services, as well as (in a later stage, during service refinement) for understanding detailed workflows in which a service plays a role, so that desired non-functional properties of services can be elicited.

Legacy System Understanding. This activity supports both identifying an IT functionality that can potentially be offered as a service, and analyzing how the envisioned service-orientation can build on the existing IT infrastructure (Arsanjani et al. 2008; Khadka et al. 2013b).

An important part of current systems are legacy systems. As succinctly expressed by Gustavo et al. (2004, p. 10), a system turns into a legacy system as soon as it is used for a different purpose or in a different context as it was originally intended for. Legacy systems are usually hard to modify (e. g., due to a lack of expertise and documentation) and expensive to maintain. However, at the same time, these systems are critical for the mission of the organization and thus, must be operational at all times (Khadka et al. 2013b; Sneed et al. 2016).

As organizations usually have different types of complex and mission critical legacy systems, it is important to consider their different (non-) functional aspects in this analysis. For instance, some legacy applications are proprietary, some are custom made. Each type poses different sets of challenges or limitations. For example, if a legacy application is a proprietary application, exposing its functionality as services may violate the terms of use (cf. Bhallamudi and Tilley 2011). We observe such (non-) functional aspects also in documented SOA migration projects. For example, Brandner et al. (2004) and Zimmermann et al. (2004), discuss a SOA migration project to create homogeneous access to functionalities offered by shared service centre for a chain of banks. Here, predominantly technical migration challenges were encountered with regards to the legacy systems, particularly when it comes to achieving interoperation between a variety of different technologies used by the service requestors of the shared service centre, e. g., Java, (D)COM, and .NET (Zimmermann et al. 2004).

Thus, we argue that a rich set of characteristics of legacy system should be considered, such as, e. g., type (proprietary, custom), age, language, interface and used data types, as well as lower-level technical aspects such as, e. g., messaging technologies or communication protocols (e. g., Lewis et al. 2006).

In the investigated SOA migration analysis approaches no specific modeling technique is recommended to foster understanding of the current system. Nevertheless, various tools and methods have been proposed to support the analysis of the legacy systems and/or existing architecture. These include: code analysis and reengineering (e. g., Ducasse et al. 2000), feature analysis (whereby features of legacy systems are extracted from code, execution traces, or user interviews, see, e. g., Millham 2010) or spectral clustering (e. g., Deiters et al. 2013). A modeling technique would form a useful complement to the mentioned legacy identification techniques and methods, in the sense that conceptual modeling allows for characterizing different IT infrastructure elements in terms of specific concepts, their attributes, and their (allowed) interrelations. For example, to express that "middleware" runs on a "mainframe server", but not the other way around; or that a software component allows for asynchronous communication, which has implications for other components trying to interact with this component. Such knowledge forms a useful analysis complement to informal IT infrastructure drawings or plain text descriptions, such as the list of features derived by the feature analysis, e. g., as done by Millham (2010).

Candidate Service Identification and Refinement. For this activity, in an initial SOA migration stage we are interested in identifying service candidates, as supported by business process analysis and service identification techniques. Subsequently, these candidate services are gradually refined into a smaller set of services that "... perform concrete functions, that have clear inputs and outputs, and that can be reused across a variety of potential service consumers. These candidate services are now specified more completely to include a definition of service inputs and outputs, and quality of service (QoS) requirements" (Balasubramaniam et al. 2008, p. 681).

In terms of modeling support, Arsanjani et al. (2008) propose service identification modeling techniques such as Goal Service Modeling, which decomposes high level stakeholder goals into an (IT) functionality being concrete enough to be offered as a service.

Target System Understanding. This activity supports understanding of the target (to-be) service-oriented design in terms of both the desired services, as well as how these services relate to the current system (Khadka et al. 2013b).

In terms of modeling, Razavian and Gordijn (2015) advocate using SoaML to express service-orientation in the target (to-be) architecture, since it expresses services exactly in terms of their relevant concepts, such as interfaces, and service contracts. However, as we point out in Sect. 3.1, modeling support for relating services to the underlying system is lacking.

Finally, at the end of the SOA analysis activity a decision on the SOA migration strategy needs to be made (Balasubramaniam et al. 2008; Khadka et al. 2013b). In brief, this concerns decisions on keeping existing applications, or to replace or outsource them (Khadka et al. 2013b), and on how the SOA migration will be planned (Balasubramaniam et al. 2008). For instance, Khadka et al. (2011) propose an approach using method engineering to determine the economical and technical feasibility of the migration based on the characteristics of legacy systems and the requirements of the target SOA application. Also decision-support methods and tools exist to facilitate selection of the migration strategy, such as the one proposed by Salama and Aly (2008). However, since enterprise modeling is not central to this phase of SOA migration planning, and a model-driven decision support on keeping existing applications arguably would require a dedicated paper, we leave this activity out of the scope of this particular contribution.

2.2 Requirements on Modeling Support

Based on the role that modeling can play in the discussed SOA migration activities, we now formulate a set of high-level requirements on modeling support for SOA migration analysis. The identified requirements are summarized in Tab. 1.

As discussed, three main aspects are important for SOA migration analysis (cf. Khadka et al. 2013a,b; Razavian and Lago 2015): (1) the as-is and (2) to-be state of the IT infrastructure, as well as (3) business concerns, which should be accounted for in each state. Therefore, the aim of the modeling approach should be to: (a) provide knowledge on the *as-is* state of the IT infrastructure with the focus on the legacy systems (cf. R1–R3 in Tab. 1) and accounting for existing dependencies and its relationship to organizational concerns (cf. R7); and (b) express the *to-be* state of the service-orientation and reflect the changes that should be performed following the selected SOA migration strategy (e. g., wrapping, re-engineering) (cf. R4–R6), and again accounting for the business concerns (cf. R7).

R1: The modeling language should allow for expressing IT landscape elements.
Rationale. For the SOA analysis activities "Legacy System Understanding" and "Target System Understanding", we are interested in expressing the observable functionality of individual IT infrastructure elements constituting an enterprise IT landscape and encompassing both software and hardware artifacts (Hanschke 2010). In this way, as stated in Sect. 2.1, we can use the language to understand the current IT landscape and identify candidate services.

R2: The modeling language should allow for expressing the dependencies between IT landscape elements.
Rationale. IT infrastructure elements can be related to each other in different ways, e. g., a software "runs on" hardware, and, at the same time, it "provides" an interface that can be used by other software to access it (Hanschke 2010). Such relationships both (1) encode in a modeling language domain "rules" as to what elements can be related and how, for example, to express that software "runs on" hardware, but not vice versa; and (2) show, for a model expressed with the language, the relevant dependencies that need to be accounted for during the SOA migration. In particular, this is relevant for analyzing how service-orientation can be built upon the current IT infrastructure (Balasubramaniam et al. 2008; Khadka et al. 2013b). For example, cf. the scenario detailed in Rosen et al. (2008): many different end-user applications can depend on the transaction

Table 1: Requirements on a conceptual modeling language supporting SOA migration

No.	*High-Level Requirement*	*Exemplary Concepts and Relationships that Should Be Accounted For*	*Supporting Sources*
	Requirements towards as-is models		
1	The modeling language should allow for expressing IT landscape elements.	database, database management system, middleware, server	Arsanjani et al. 2008; Gustavo et al. 2004; Khadka et al. 2013b; Sneed et al. 2016
2	The modeling language should allow for expressing the dependencies between IT landscape elements.	uses, provides, runs on	Arsanjani et al. 2008; Balasubramaniam et al. 2008; Khadka et al. 2013b; Rosen et al. 2008
3	The modeling language should account for characteristics of legacy systems as well as for legacy system dependencies.	mission criticality, source code availability, implementation language, code complexity, availability of documentation, strength of support	Bhallamudi and Tilley 2011; Khadka et al. 2013a; Lewis et al. 2006; Razavian and Lago 2015
	Requirements towards to-be models		
4	The modeling language should provide dedicated concepts that allow to model a concept of service and its relevant types.	IT service, web service, interface	Arsanjani et al. 2008; Razavian and Gordijn 2015
5	The modeling language should allow for relating a service to its underlying implementation, in accordance with the migration strategy used.	wrapps, provides, runs on, uses	Balasubramaniam et al. 2008; Gustavo et al. 2004; Khadka et al. 2013b
6	The modeling language should account for quality attributes of services.	QoS attributes, e. g., response time, throughput, availability, reliability, average execution time	Arsanjani et al. 2008; Razavian and Gordijn 2015 Rosen et al. 2008, pp. 94–98
	Accounting for business-IT alignment in as-is models and to-be models		
7	The modeling language should allow for expressing dependencies between the IT landscape and the organization action system, encompassing, among others, business processes, organizational structure as well as business goals.	e. g., supports – to relate IT infrastructure elements with business processes or business goals, accountable for – to relate IT infrastructure elements with the organizational structure	Balasubramaniam et al. 2008; Khadka et al. 2013a; Papazoglou and Heuvel 2006; Rabelo et al. 2015; Rosen et al. 2008
	General Requirements		
8	The modeling approach should support the productivity of modeling and analysis.	to support productivity and enable various analyses, semantically rich concepts are needed, cf. Frank (2013)	Bhallamudi and Tilley 2011; Frank 2013; Khadka et al. 2014, 2013a,b; Lewis et al. 2006; Razavian and Lago 2015
9	The modeling approach should provide a corresponding tool support that would facilitate the design and analysis process.	n.a.	Frank 2013; Khadka et al. 2014

processing capabilities of a mainframe, implying that if a change is made to the mainframe (e. g., to expose the transaction processing capabilities via web service technology), then the end-user applications relying on the mainframe need to be modified accordingly.

R3: The modeling language should account for characteristics of legacy systems as well as for legacy system dependencies.
Rationale. Concerning the SOA migration analysis activities "Legacy System Understanding" and "Target System Understanding", for any meaningful analysis on the possible behavior of IT infrastructure elements, we need to consider their

characteristics (Razavian and Lago 2015). To illustrate this, recall here our discussion of exemplary legacy system properties in Sect 2.1, such as the programming language used, or whether a legacy system is proprietary or custom made. These properties influence how one interacts with the legacy system by, for example, having to invest an additional effort to abstract away from the used programming language, or to avoid violating terms of use in case of a proprietary legacy application. In addition, please note that the required level of details depends on the migration strategy followed: the black-box strategy requires information on the functionality offered by the legacy system and how to access it, whereas the white-box strategy requires detailed information about the inner workings of a legacy system, going down to code characteristics and its availability (Comella-Dorda et al. 2000).

R4: The modeling language should provide dedicated concepts that allow to model a concept of service and its relevant types.
Rationale. For the SOA migration analysis activities "Target System Understanding" and "Service Identification and Refinement" the language should provide (rudimentary) expressiveness for service-orientation. First and foremost, this entails that it should provide concepts to express a service. Three features of services are considered to be crucial (Bouguettaya et al. 2017, p. 70): functionality (specified by the operations offered by a service), behavior (how the service operations can be invoked), and quality, which determines the non-functional properties of a service (we will discuss the latter in R6). Therefore, we need to account for concepts to describe the functionality offered by a service as well as, among others, its operations and how the service is to be accessed (Razavian and Gordijn 2015).

R5: The modeling language should allow for relating a service to its underlying implementation, in accordance with the migration strategy used.
Rationale. Often, a service-orientation complements elements of an existing IT infrastructure instead of replacing them (Arsanjani et al. 2008; Khadka et al. 2013b). For example, a web service can be used to expose existing transaction processing capabilities of a mainframe via a standardized interface (cf. Rosen et al. 2008). Therefore, to provide a meaningful analysis of how a service builds upon, and interacts with, the current IT infrastructure, modeling support should show how to relate service concepts to elements of the existing IT infrastructure that are necessary for realizing the desired functionality.

R6: The modeling language should account for quality attributes of services.
Rationale. Expressing quality attributes of services (such as throughput, response time, and availability) is relevant for both activities "Target System Understanding" and "Service Identification and Refinement".

For the SOA migration analysis activity "Service Identification and Refinement" the quality attributes are relevant for the specification of candidate services. In particular, according to Balasubramaniam et al. (2008, p. 681), during service refinement, candidate services are gradually refined into a smaller set of services that "include a definition of service inputs and outputs, and Quality of Service (QoS) requirements".

For the SOA migration analysis activity "Target System Understanding", we desire to know such quality attributes because they constrain how one uses the functionality offered by a (web) service (D'Ambrogio 2006), for example, in terms of how quickly a request is responded to, or how quickly a transaction is processed.

R7: The modeling language should allow for expressing dependencies between the IT landscape and the organization action system.
Rationale. In line with the SOA migration analysis activity "Candidate Service Identification and Refinement" (see Sect. 2.1), a SOA migration analysis focuses not only on the IT infrastructure itself, but equally on a business perspective. Business processes largely drive what is implemented in terms of IT support (Papazoglou and Heuvel 2006; Rabelo et al. 2015), and vice versa:

IT infrastructure characteristics, such as, e. g., the average processing speed of a request by a web service, influence what is possible from a business point of view (Papazoglou and Heuvel 2006). As such, this imposes the requirement that any conceptual modeling support should consider service-orientation from multiple perspectives, so as to be able to analyze business and IT concerns in tandem (Bucher et al. 2006).

R8: The modeling approach should support the productivity of a modeling and analysis process.
Rationale. So far, we have discussed several aspects of SOA migration analysis that hint at a rich variety of domain concepts and domain "rules". This includes quality of service attributes of a web service (e. g., the average processing speed), which influence what is possible from a business point of view, or attributes of legacy system elements (such as the used programming language) which determine how one subsequently interacts with them.

In designing our language in support of SOA migration analysis, we desire to capture this rich variety. Therefore, there is a need to emphasize modeling productivity (cf. Frank 2013) over reuse: we aim to provide analysts with a language that encodes already most relevant domain concepts and domain "rules", so that the analyst does not have to reconstruct such concepts from scratch. Instead, s/he can (1) create models that are in line with the rules of the domain. For example, stating that a piece of software runs on a piece of hardware, but not the other way round; and (2) use concepts that are readily available to support detailed relevant analyses. For example, we require an explicit consideration of QoS attributes for web services when it comes to assessing how business processes interact with these web services.

Of course, we are aware that we cannot capture every detailed domain characteristic into our language since our language would simply grow out of control, especially when increasing the number of analysis scenarios it is supposed to support. Rather, we opt for a trade-off between language productivity and reuse (cf. Frank 2013): we want to make sure that our language contains sufficient semantic richness to support an analyst, yet not at the expense of considering every detail. For example, while it can be relevant for very particular application scenarios, we do not capture that the size of data invocations/replies of CICS (a particular type of legacy application) transactions is a maximum of 32 bytes (Rosen et al. 2008, p. 373).

Note that we return to the balancing of reuse and productivity in the conclusion, where limitations of the currently used modeling language paradigm are discussed.

R9: The modeling approach should provide a corresponding tool support.
Rationale: In order to support cross-aspect analysis and support productivity of the modeling and analysis process, we require a corresponding software tool (Khadka et al. 2014). The application of each modeling method significantly benefits from the availability of a corresponding modeling environment that guides the modeling process and avoids the creation of syntactically incorrect models by performing automated syntax checks. The tool can also support automated analyses, thus, facilitating SOA migration analyses on created models.

3 Existing Approaches and Fulfillment of Requirements

The relevant modeling approaches can be roughly divided into approaches for modeling services and SOA, standalone languages for modeling business and IT concerns, and enterprise modeling (EM) approaches. These three groups of approaches are discussed subsequently.

3.1 Service and SOA Modeling

Different service and SOA modeling approaches exist, some as stand-alone modeling languages, some in support of comprehensive SOA migration methods. An example of a stand-alone language is SoaML (OMG 2012), which is an OMG standard language for analyzing service-orientation, with a basic capability of linking business aspects

and IT aspects (OMG 2012, p. 7). SoaML focuses on service-oriented concepts, such as the interfaces and contracts between SOA participants. By focusing on service-orientation, SoaML has proven useful in practice. For one, by applying SoaML to the health-care industry (Silva et al. 2015), authors compare SoaML favorably to using (generic) UML class diagrams by pointing out that SoaML offers concepts relevant for expressing service customers and service suppliers. However, by focusing on service-orientation, SoaML purposefully abstracts away from the realization of services in terms of the underlying infrastructure. Furthermore, SoaML is a UML profile and, as such, lacks a dedicated concrete syntax for expressing its concepts. Also SoaML provides no concrete conceptualization of QoS attributes.

Further consider SOMA-ME, which is a design framework that supports the model-driven design of SOA solutions using the SOMA method (Zhang et al. 2008). As a part of SOMA-ME, an integrated development environment is provided that builds on the IBM Rational software development platform. It uses UML as the language for modeling software systems. Thus, UML profiles are provided to capture information about services, such as origins, associated goals, key performance indicators (KPIs), non-functional requirements, and interdependencies among services. In SOMA-ME, the profiles are contained in two packages: the SOA method-profile package and the SOA architecture-profile package (Zhang et al. 2008). Considering the goals of this paper, only the first package is relevant for our further discussion. This package includes the following aspects: SOA business processes profile (containing stereotypes to support service identification techniques), SOA services profile (containing stereotypes that address the service identification and service specification steps), and SOA service components profile (capturing information related to service realization). However, like SoaML (a) SOME-ME lacks a dedicated visualization, it being largely built as a UML profile, and (b) SOME-ME lacks a rich sets of attributes to, for example, express QoS.

The Service-Oriented Modeling Framework (SOMF) is a service-oriented life cycle modeling method proposed by Bell (2008). According to it, all software assets can be subject to modeling activities and are seen as services, i. e. service-oriented modeling elements. Bell (2008) proposes concepts for expressing service-orientation, such as “atomic service” versus “composite service”. Also, he includes a link between business services and the IT services needed to realize these. However, Bell (2008) proposes only a set of concepts. Relationships between concepts, constraints, and attributes are only loosely defined, if at all. For one, possible constraints are limited to plain text guidelines for using the concepts, that are moreover not always clear or well explained. For instance, a guideline can be “only business domains can be layered” (Bell 2008, p. 159), without expressing it within a language specification.

Next, the Topology and Orchestration Specification for Cloud Applications (TOSCA) (OASIS 2013), an OASIS standard language, provides a service template to define the structural and management-related aspects of IT services. TOSCA aims to standardize the description of software applications that run in the cloud. Thus, it offers the core concepts required to describe an application, as well as its dependencies and supporting (cloud) infrastructure (Brogi et al. 2014). However, the key TOSCA concepts are generic. Namely, there are two basic concepts: nodes and relationships. Both can be extended, e. g., by adding specific Node Types and Relationship Types (Brogi et al. 2014), in order to describe a specific situation. However, due to its reliance on a small set of generic concepts, natively TOSCA does not support the modeling productivity that we target (cf. R8).

In turn, Terlouw and Albani (2013) apply the way of thinking of DEMO (Design and Engineering Methodology for Organizations, cf. Dietz 2006) to conceptualize services for the needs of service-orientation. They provide a formal description of a service in terms of business issues, such as the interaction between customer and supplier, as specified by the underlying service

contract. However, in line with DEMO, the service description is largely implementation independent (Terlouw and Albani 2013). As such, no details are provided about the underlying IT infrastructure. Note that, like SoaML, DEMO has also been successfully applied in various practical settings. Among others, the DEMO-way of thinking applied to service-orientation has proven interesting for an insurance company, which in particular appreciated making explicit (linguistic) coordination acts in specifying a service (cf. Terlouw and Albani 2013, pp. 98-99).

Finally, regarding detailed IT infrastructure concerns as linked to service-orientation, Fuhr et al. (2013) and Khadka et al. (2011) propose methods to convert code of legacy systems into code for SOAs. To this end, they combine capabilities of existing SOA design methods (e. g., SOMA, Fuhr et al. 2013), with model-based techniques from Model-Driven Engineering/Model-Driven Architecture (MDE/MDA) (e. g., a graph-based mechanism, Fuhr et al. 2013) to allow for code transformation. While both Fuhr et al. (2013) and Khadka et al. (2011) deal with detailed IT infrastructure concerns (namely legacy systems) and use model-driven techniques, they do not provide a model-based support analysis of IT infrastructure per se. Rather their purpose is to use model-driven techniques for code transformation.

3.2 Standalone Modeling of Business and IT Aspects

Various (standalone) modeling approaches exist which support focused understanding of selected business related aspects and IT aspects. Among them are approaches targeting at modeling of goals and motivation, e. g., i* (Yu 1997), Tropos (Bresciani et al. 2004), KAOS (Dardenne et al. 1993) or the Business Motivation Model (cf. Sect. 2.1); modeling of business processes, e. g., by means of the Business Process Model and Notation (BPMN) (OMG 2011); or rudimentary modeling of IT infrastructure by means of deployment diagrams (OMG 2015c, p. 651).

Each of these languages comes with different sets of concepts and analytic capabilities, which, as already mentioned in Sect. 2, can be used within one of the mentioned SOA migration activities. However, as these standalone modeling approaches focus on selected aspects of an enterprise only, they do not allow for a more comprehensive, integrated analysis accounting for multiple perspectives.

This also applies to deployment diagrams. Even though they are part of the comprehensive Unified Modeling Language (UML) (OMG 2015c), for UML a primary unit of concern is *software*, and not legacy systems or service-orientation as such. Therefore, its support for SOA migration projects is limited.

3.3 Enterprise Modeling Approaches

Unlike standalone modeling approaches, Enterprise Modeling (EM) approaches usually cover *multiple* perspectives on an organization (e. g., by considering in tandem organizational goals, business processes, or IT infrastructure), and relate these perspectives to each other. Exemplary approaches include ArchiMate (The Open Group 2013), Architecture of Integrated Information Systems (ARIS) (Scheer 2001; SoftwareAG 2017), 4EM (Sandkuhl et al. 2013), and Multi-Perspective Enterprise Modeling (MEMO) (Frank 2012).

Here, we focus on EM approaches that can express facets that are of particular interest for a SOA analysis. In line with our requirements (see Tab. 1), the EM approach should express IT infrastructure concerns, service-orientation, business context, and various interrelations among those aspects. Only three EM approaches consider IT infrastructure modeling explicitly and, at the same time, account for the service-orientation: ArchiMate (version 2.1, which includes the motivation extension), MEMO with the IT Modeling Language (ITML) (Heise 2013) and ARIS v10.

ArchiMate is an open enterprise architecture modeling standard offering different viewpoints on enterprise architecture (Business, Application, Information, Technology) (The Open Group 2013). ArchiMate's goal is to define a modeling language for "the representation of enterprise architectures [...] as well as their motivation and rationale" (The

Table 2: The selected concepts of EM approaches supporting modeling of an IT Landscape - an overview

Approach	*Software*	*Hardware*	*Relationship types*	*Attributes*	*Constraints*
ArchiMate v2.1 (The Open Group 2013)	Artifact, Node, SystemSoftware, ApplicationComponent, ApplicationInterface, ApplicationService, InfrastructureService	Node, Device	Access, association, used by	-	-
MEMO ITML (Heise 2013)	Software, Architecture, License, SoftwareProduct, Service, SoftwareInterface, ComunicationProtocol, Data	Computer, PhysicalDataMedium, NetworkComponent, Network, Printer, Scanner, Fax, HardwareDeviceRole	A set of domain relationships, e. g., used in, requires, runs on, refers to	A set of attributes defined for each concept	A rich set of OCL-based constraints
ARIS v10 (SoftwareAG 2017)	IT system, Domain, Application system, Application system type, Module type, Interface, Protocol, Cluster, Service capability, Business service, Software service type, IT function type	Hardware, Component, Network, Network Component	Assignment, connection	Name, description, definition, author	

Open Group 2013, p. 2). This is visible in ArchiMate's intention to "not [...] introduce a language that can replace all the domain specific languages that exist" (Lankhorst 2013, p. 77) and to offer generic concepts for describing main elements of different organizational domains and relationships between them (Lankhorst 2013, pp. 76-77).

In turn, the stated key goal of MEMO is to integrate different aspects that should be considered while designing, implementing and using business information systems (Frank 1994). MEMO offers a set of integrated Domain Specific Modeling Languages (DSMLs), such as languages for modeling business processes and organizational structures (OrgML, Frank 2014), goal modeling (GoalML, Overbeek et al. 2015) and, especially relevant for our purposes, the Information Technology Modeling Language (ITML) for IT infrastructure modeling (Heise 2013). As those DSMLs are integrated, IT infrastructure models can be related to organizational concerns, which fosters communication between stakeholders with different professional backgrounds, and allows for cross-perspective analyses (Heise 2013).

ARIS offers a high-level framework (the so-called "House of Business Engineering", Scheer 2001) together with a large set of diagrams. It provides, among others, an original modeling language, the "Event-driven Process Chain" (EPC), and refers to existing modeling languages such as ERM, DFD, or BPMN. The method is supported by a comprehensive commercial toolset, which currently offers an extensive set of diagram types to capture characteristics of IT infrastructure, such as the application system type diagram, the application system diagram, the program flow chart, the network diagram, or the access diagram (SoftwareAG 2017).

Although these approaches exhibit similarities (cf. Bock et al. 2014), they are substantially different. For one, there is a substantial difference in the semantic richness of the modeling concepts. While ArchiMate and ARIS favor a concise language design by focusing on a small set of essential enterprise (architecture) concepts, MEMO provides domain stakeholders with elaborate reconstructions of the (technical) concepts that they are familiar with. Thus, the languages support different analysis scenarios. Particularly, while ArchiMate, ARIS and MEMO offer means to describe IT infrastructure together with service-orientation (cf. Tab. 2), they do so at differing

levels of granularity. And so, ArchiMate provides a set of generic concepts with no attributes and constraints. Similarly, although ARIS offers an extensive set of diagram types, its individual diagram types offer generic concepts with few attributes and relationships. For one, in terms of diagrams ARIS differentiates between an application system diagram and application system type diagram, so as to distinguish between analyses pertaining to the running system versus analyses pertaining to logical dependencies in the system. However, in these two diagrams ARIS offers only a limited expressiveness, e. g., by offering the high-level concepts "application system type" and "module type", with a limited number of attributes and relationships for each. In contrast, MEMO ITML offers a set of more fine-grained concepts with a rich set of attributes.

Differences are also visible in the other areas covered by the approaches. For one, MEMO offers a dedicated DSML for business process modeling (OrgML), which provides both conventional business process concepts (e. g., ControlFlowSubProcess, Event, Task) as well as more elaborate ones (e. g., an Exception). Differently, by design ArchiMate provides a limited number of business process modeling concepts (Business Process, Business Event). Also for goal modeling, MEMO again offers a dedicated DSML, GoalML, which defines a comprehensive set of concepts allowing to express various aspects of enterprise goals (e. g., EngagementGoal, SymbolicGoal, GoalConfiguration, cf. Overbeek et al. 2015), all with a rich set of attributes. Differently, the motivation extension of ArchiMate offers concepts such as Goal, Principle and Driver, but these remain underspecified up to the point that they are difficult to tell apart by language users (Engelsman and Wieringa 2014).

Also it is noteworthy that Ziemann et al. (2006) applies ARIS to support service modeling, in particular for the business-driven analysis for the needs of web service development. Namely, the authors rely on EPC business process models to identify how web services integrate into the processes of an organization. However, this work can be perceived as preliminary, because the authors use only business process models and disregard the other elements of the enterprise action system and information system. For instance, they did not relate the business process models to organizational goals, nor to the organizational structure, or to the overall enterprise IT Infrastructure.

Finally, of note is that (by design) none of the discussed EM approaches offers dedicated concepts for expressing specific aspects of legacy systems or SOA migration projects, such as wrappers or middleware.

3.4 Fulfillment of Requirements by the Discussed Approaches

Tab. 3 provides a short summary of the evaluation of the discussed approaches. In order to assess whether the discussed approaches fulfill the identified requirements, we have analyzed the available documentation of those and contrasted it with, among others, candidate concepts and relationships. A subset of these candidates is provided in Tab. 1. Please note that we are primarily interested in the productivity of modeling and semantically rich concepts (cf. R8). This implies that if a considered language offers a set of concepts (like, e. g., a "node" in TOSCA) that can be used to define the element of interest from scratch, we do not consider the relevant requirement as fully fulfilled. Indeed, we consider the given requirement to be fully fulfilled, only if a language specification as such is already providing the required abstractions, so that they do not have to be recreated anymore from scratch.

The first part of Tab. 3 summarizes the discussed service/SOA modeling languages in terms of their fulfillment of the identified requirements. Observe that, while the existing service modeling approaches (such as SOMA-ME) provide conceptualizations of a service and a (rudimentary) relationship to business concerns, they notably lack a capability to express existing IT infrastructure and its relationship to service-orientation. Even if such a capability is offered by providing a set of generic, extensible concepts (cf. TOSCA), a substantial effort is required in order to define all required aspects. For example to encode basic

Table 3: Summary on the requirements fulfillment by the main discussed approaches

Approach	*R1*	*R2*	*R3*	*R4*	*R5*	*R6*	*R7*	*R8*	*R9*
SOMA with SOMA-ME	○	○	○	●	◑	◑	○	◑	◑
SOMF	○	○	○	●	○	◑	◑	◑	○
SoaML	○	○	○	●	○	○	◑	◑	○
TOSCA	◑	◑	◑	◑	◑	◑	◑	◑	●
DEMO-based Service Concept	○	○	○	●	○	◑	●	◑	○
ArchiMate with motivation exten.	◑	◑	○	●	●	○	●	◑	◑
MEMO	●	●	●	◑	◑	◑	●	●	●
ARIS	◑	◑	◑	●	◑	○	●	●	◑

Legend: ○– not covered; ◑– partly covered; ●– largely covered

"domain rules" such as software *runs on* hardware, but not vice versa.

The standalone modeling approaches, as already mentioned in Sect. 3.2, focus on selected aspects of an enterprise only, and thus, they do not allow for a more comprehensive, integrated analysis accounting for multiple perspectives. Although it is of course possible to select a few standalone approaches (such as BPMN, or KAOS), their integration would not be straightforward. This is mainly because those languages are not necessarily compatible: each approach is based on different modeling assumptions and a different language architecture (Bock et al. 2014), thus, using them in tandem and being able to conduct cross-language analyses would require moving them to one language architecture/one paradigm. In addition, a corresponding tool would need to be created, and/or a tooling chain needs to be catered for. Therefore, the standalone modeling approaches do not support the formulated requirements and thus, are not accounted for in Tab. 3.

Finally, although various enterprise modeling approaches exist that allow for modeling IT infrastructures and/or expressing a service-orientation from various angles (OMG 2012; Terlouw and Albani 2013; The Open Group 2013, to name a few), these approaches often on purpose forgo the level of detail that is required to analyse IT infrastructure for the needs of SOA migration. In addition, as we detail in Sect. 3, often they insufficiently relate IT concerns and organizational concerns.

As shown in Tab. 3, ArchiMate, ARIS and MEMO offer (1) concepts for expressing IT infrastructure (R1–R3), (2) the possibility to express service-orientation (R4–R6), (3) linking of business concerns and IT concerns (R7). Nevertheless, there is a difference in the extent to which MEMO, ARIS and ArchiMate provide model-driven support to SOA migration projects. In particular, ArchiMate and ARIS lack expressiveness for IT infrastructure concepts that we would need to meaningfully analyze requirements for moving towards a service-oriented IT infrastructure. First, consider ArchiMate. By design ArchiMate offers a concise set of coarse-grained IT infrastructure concepts – just enough for expressing enterprise architecture concerns. However, due to their abstract nature these concepts are too multi-interpretable for a meaningful domain-specific analysis. Furthermore, ArchiMate concepts lack attributes, which limits their semantic richness and hence, suitability for various detailed domain-specific analyses.

This similarly holds for ARIS. Although a set of concepts is offered that allow to express software and hardware related aspects of an IT infrastructure, the concepts are still generic (e. g., "module" for the application system type diagram). Furthermore, although the concepts as such are equipped with attributes, those attributes are limited and

do not allow to express required information. In addition one does not have an access to ARIS meta model, and the corresponding software tool is proprietary. As a result of the restricted access, one cannot extend the set of concepts and attributes of the ARIS meta model, and neither make the necessary corresponding tool modifications (hence the "partly covered" score on the tool support requirement (R9) in Tab. 3 for ARIS: while a tool is available, it is proprietary and therefore, not extendable).

In contrast, the language architecture followed by MEMO offers an extensive set of IT infrastructure concepts as a representation of the universe of discourse of IT experts. All of the concepts are equipped with a rich set of attributes, and thus, also account for their non-functional aspects. However, ITML has not been designed to support SOA migration projects (Heise 2013; Kirchner 2008). As such, ITML does little in the way of expressing legacy systems and service-orientation. As a result, to use ITML for a SOA migration analysis, we require additional concepts and attributes. Therefore, as shown by the fulfillment of the identified requirements (cf. Tab. 3), extensions are needed to MEMO ITML so that it is able to fully support the SOA migration analysis.

A critical success factor for extending an existing modeling approach is that its specification is both, precise and complete, as well as publicly and freely available. As all of the MEMO language specifications are available to interested audience and in addition, the introduced changes can be reflected in the current software tool supporting MEMO, we deem it to be more suitable for extensions compared to ARIS.

To summarize: in the light of the above discussion and conducted analysis, we deem MEMO to be the most promising candidate. However, to make ITML suitable for our purposes, we extend it to provide support for SOA migration analysis.

4 Extended MEMO ITML

As already discussed in Sect. 3.3, ITML is part of a comprehensive method for multi-perspective enterprise modeling MEMO (Frank 2012). We extend the MEMO ITML in order for it to better support analyses in the context of SOA migration projects. In addition, by doing that we can benefit from the already developed other MEMO DSMLs focusing on the selected aspects of an enterprise action system.

4.1 MEMO MML and Modeling Decisions

Several means of defining a modeling language exist, however, the one frequently used, also in case of MEMO, is by specifying a meta model, i. e., a model of models. A meta model defines the abstract syntax and semantics of a given language together with additional constraints (Frank 2011, p. 3). Thus, a model (the M_1 level) is specified by a modeling language, which in turn is specified by a meta model (the M_2 level). Constraints are usually formulated by using the Object Constraint Language (OCL) (Warmer and Kleppe 2003).

As we extend the MEMO ITML, we use the MEMO method's common Meta Modeling Language (MML) (Frank 2011), which allows for integrating the *extended* ITML into the MEMO method's language architecture (Frank 2012, pp. 947-950). Consequently, this allows us to use the *extended* ITML in tandem with other MEMO languages.

When compared to "traditional" meta modeling languages, MML provides additional language constructs for expressing: (a) intrinsic attributes and relationships, and (b) language level types. Intrinsic attributes and relationships are instantiated only on the instance level and not on the type level, and are visualized with a white letter "i" on a black background. In turn, language level types are instantiated on the type level only, no further, and are visualized with a grey background of the concept's name (Frank 2011, pp. 23-24).

Defining a meta model implies making modeling decisions, such as: if a concept should be part of the language specification (meta type, M_2 level) or part of the language application (type, M_1 level). To support making design decisions and to ensure the required quality of the developed meta

model, the guidelines proposed by Frank (2013) have been applied to extend and enrich the previous version of ITML with additional concepts, attributes and relationships (cf. Fig. 2).

4.2 Extended Abstract Syntax

Fig. 2 presents a fragment of the extended ITML meta model. It also highlights the modifications to the original ITML meta model (Heise 2013) by adding colored squares to the upper right corner of added/modified concepts, colored squares to definitions of new relationships, and transparent boxes over added attributes (cf. the legend, Fig. 2). For comprehensibility purposes, we structure the discussion of the meta model in three parts: (1) "software"-related concepts, and the functionality offered by this software. This is largely relevant for identifying characteristics of legacy systems, and the functionality offered by such legacy systems; (2) selected specializations of software, e. g., "middleware" and "wrapper" concepts, that are relevant to enable interoperation between heterogeneous system elements; (3) "service-oriented" concepts, i. e., we discuss here concepts to express service-orientation, such as web services.

As, in line with R8, our aim is to support the productivity of modeling as well as analysis, each of the modeled concepts has a rich set of domain-specific attributes and relationships. However, in order for the proposed language to be still usable in case relevant changes occur (e. g., a new type of software system is introduced), the generic meta types, such as "Software" have not been made abstract, thus, they can be instantiated to model newly identified phenomena whenever necessary.

In the following, we discuss selected design decisions, which are relevant for the needs of our further discussion.

Software and its main relationships

The concept "Software" captures any software artifact that provides some functionality. It is further specialized into different categories of software artifacts to account for the different roles software can play in the enterprise IT landscape (Hanschke 2010), such as application software, operating system, or otherwise.

Extended ITML expresses both functional and non-functional aspects of "Software". Regarding the former, firstly, the extended ITML offers the concept of a "Topic" (defined as a Language Level Type) with its specializations "DataTopic" and "Function", cf. Fig. 2. A function is understood as a task, a process or an activity that is intended to achieve one or several (business) goals. By taking advantage of a "Function" (for example, a function "financial report generation"), it is possible to state the functionality offered by any "Software" artifact, which is particularly helpful for identifying candidate services. Please note that, analogous to the function trees from ARIS (SoftwareAG 2017, p. 14), "Functions" are organized in hierarchies (by using the "partOf" relationship), thus, it is possible to define the function topics on different levels of granularity. Finally, "Functions" can be related to the defined goals and processes of an organization (via the relationship to the "SpecificSupport" concept).

Furthermore, the extended ITML offers added expressiveness regarding the software "Interface" so as to enhance analysis capabilities with respect to how to access it. In particular, this concerns (1) the addition of a relationship "described_with", which allows to relate the interface to the corresponding language level type "Language", to express what language the interface is specified with (this language is also called the interface description language); (2) the specification of particular parameters with the auxiliary type "Parameter". This constrains the format of interface inputs and outputs, compared to an earlier version of ITML (Heise 2013), which expressed inputs and outputs as strings (thus, allowing for anything).

Also the "CommunicationRelationship" has been enhanced with additional attributes (e. g., regarding messaging and transportation protocols), information about the exchanged format (cf. the corresponding language level type) and, if applicable, the middleware configuration required for a communication.

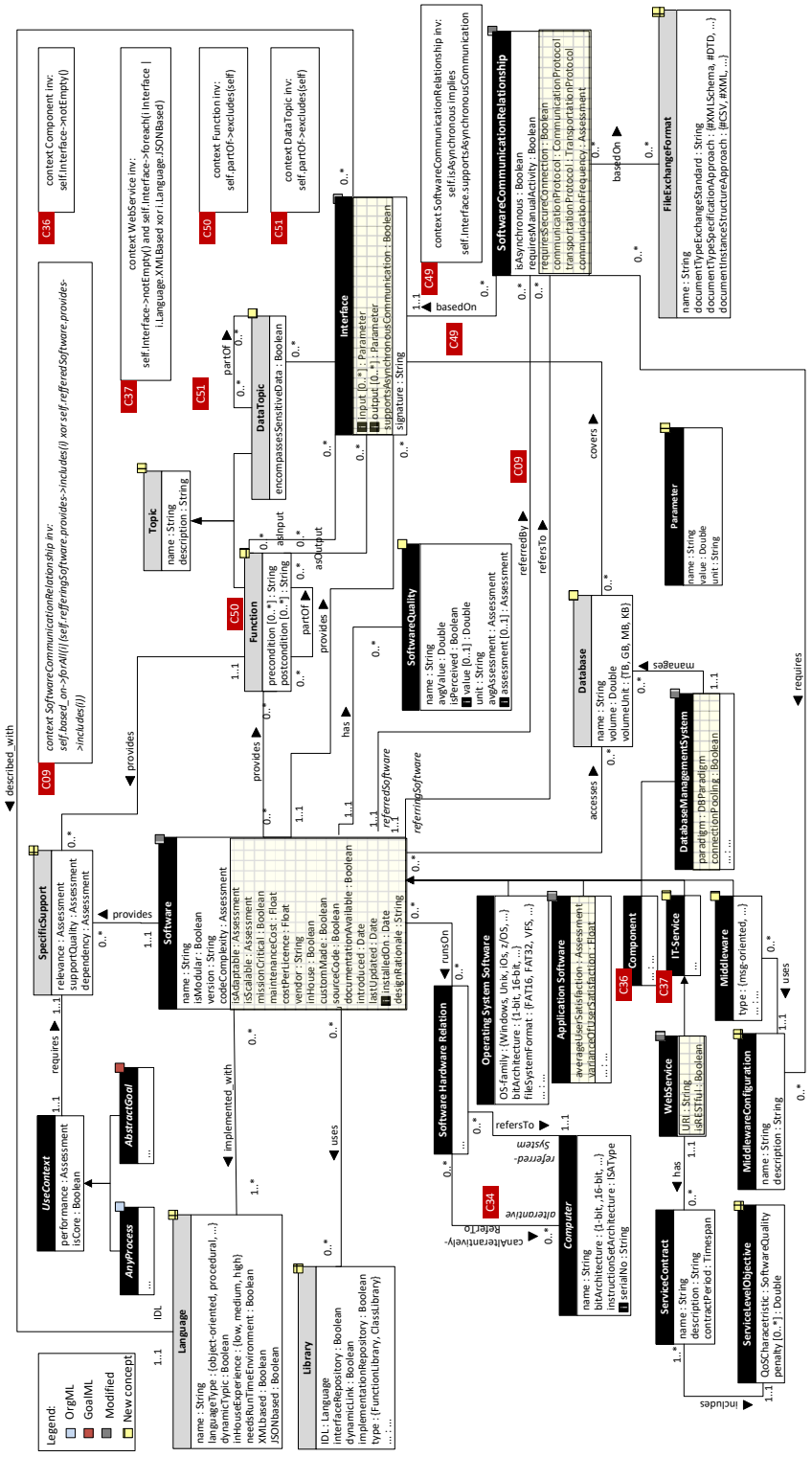

Figure 2: An Excerpt of the Meta Model of the Extended ITML

Moreover, we added the "Library" concept. A library is a collection of resources that are written using an implementation language, and that are used by some software. A "Library" has a well-defined interface description language by which the behavior is invoked. A "Library" may be used by some "Software".

For the non-functional aspects of software, to aid further in legacy system understanding, we added a set of attributes to the "Software" concept. Selected examples follow. A dedicated attribute has been added allowing to express whether the software is commercial off-the-shelf (COTS) or in-house, which is largely relevant since, as per Sect. 2, exposing the functionality of proprietary applications may violate their terms of use. Secondly, we added attributes to express user-based assessment of different aspects of either a service or a software (partly accounted for in a "Software", partly in "ApplicationSoftware", partly in an "IT Service"). Such user based assessments include (a) the possibility to flag (legacy) software as mission critical, thus, indicating its particular importance to the organization, as well as (b) various assessments that help to understand the complexity of modifying the software in line with the migration, by, e. g., assessing its scalability (through the attribute "isScalable") or adaptability (through the attribute "isAdaptable").

Finally, we consider the relationship between the IT landscape and the enterprise action system (cf. R7). To model such relationships, we use two concepts: "SpecificSupport" and "UseContext" (cf. Fig. 2). "SpecificSupport" allows for relating the IT landscape concepts, specifically "Software", to organizational concerns. Observe that "SpecificSupport" has a set of non-functional attributes, which allow to annotate the relationship between ITML concepts and concepts from other MEMO DSMLs, being specializations of "UseContext". Thus, "UseContext" allows for connecting a "SpecificSupport" to a key concept of any of the organizational DSMLs that are part of the MEMO family, e. g., a concept "BusinessProcess" defined within OrgML (Frank 2011) or "AbstractGoal" specified in GoalML (Overbeek et al. 2015).

Middleware and wrapper concepts

Middleware and wrapper are important to achieve interoperation amongst heterogeneous systems. "Middleware" refers to programming abstractions (cf. Gustavo et al. 2004, p. 30) that hide (some of) the complexity involved in building distributed software applications.

To account for different ways one interacts with middleware, as can be seen in Fig. 3 we distinguish between four middleware types (Gustavo et al. 2004, pp. 32–40): Remote Procedure Call (RPC)-based middleware, message-oriented middleware, transaction processing middleware, and object-oriented middleware. RPC-based middleware provides programming abstractions to make a call to a remote procedure appear as if it were a local procedure call, thus, allowing one to abstract away from networking details and the internal details of the called procedure. Message-oriented middleware, transaction-based middleware and object-oriented middleware build on the same ideas as RPC-based middleware, while adding additional programming abstractions to respectively account for (1) message-orientation. The basic idea behind message-orientation is that distributed applications communicate with each other *a-synchronously*, by exchanging messages (messages are structured datasets, cf. Gustavo et al. 2004, p. 60). Typically the message-oriented middleware enables such asynchronous communication by providing message queues, which store messages placed there by a client, so that a receiving application can pull messages from the queue *when it is ready to do so*. This is opposed to RPC-based middleware, whereby the distributed applications have to be constantly listening for calls (Gustavo et al. 2004, p. 62); (2) transactions: roughly speaking, this means that in addition to individual procedures one needs mechanisms to account for collections of procedures to be executed, and their ordering, as part of a transaction (e. g., to account for receiving a "reply" after a "request"); (3) object-orientation, which means

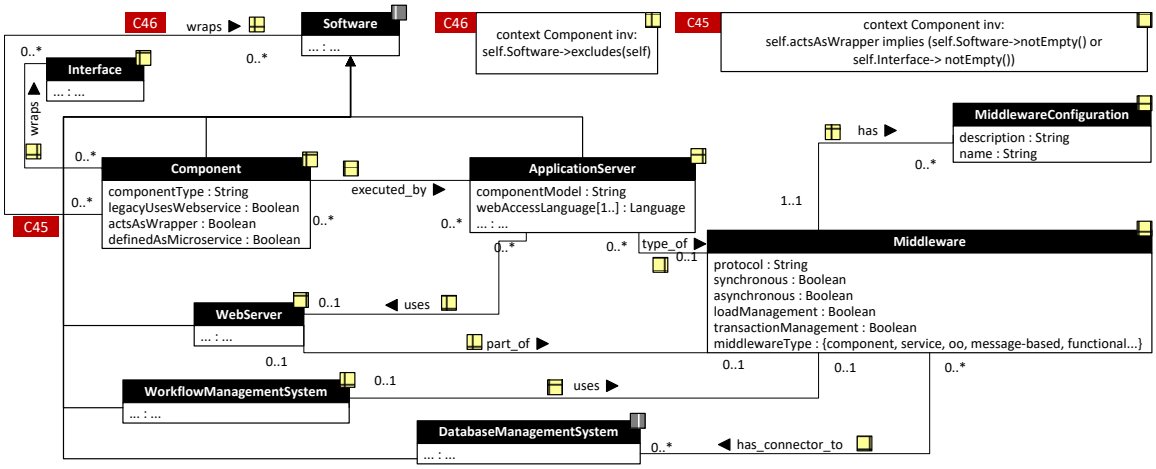

Figure 3: Extended ITML metamodel: Middleware and wrapper concepts

that, roughly speaking, one provides additional mechanisms for accessing remote objects. An example mechanism is encapsulation (Gustavo et al. 2004, p. 57), which hides the internal details of objects being exchanged, such as the used programming language.

Middleware can be part of application server, or a web server. While an "ApplicationServer" and "WebServer" act very much like conventional middleware, they have one key difference: web access capabilities (Gustavo et al. 2004, p. 103). Such web access capabilities are prominently visible in the form of web access languages supported by the application/web server, such as HTTP.

Various relationships exist between middleware and other software, of which we exemplary modeled two: a workflow management system "uses" middleware to enable interoperability amongst various possibly heterogeneous components of the workflow system (Gustavo et al. 2004, p. 89), and middleware "has_connector_to" a database management system, to emphasize that middleware solutions allows different types of software database access, in terms of submitting a query and retrieving results (Rosen et al. 2008, pp. 375–376).

Whereas a middleware can provide a complex set of functionalities for enabling interoperation, a "Wrapper" is a relatively basic type of software. Their role is to allow for abstracting away from the inner workings of legacy systems, focusing on their inputs and outputs instead. Wrappers take on a particularly important role when following a black-box migration strategy, see Sect. 3. Indeed, wrapping components provide several features to expose parts of a legacy application (data, logic, interface) and assist in encapsulating the implementation details of the required service, defining a descriptive interface and publishing the service in a discoverable location (Baghdadi and Al-Bulushi 2015). Thus, by hiding the internal workings of a single (legacy) system, as detailed by Baghdadi and Al-Bulushi (2015), they offer the toolset to expose a legacy system in an interoperable manner.

To account for a wrapper, we introduce it as a type of software component, and further detail it by means of (1) various "wraps" relationships to different types of software that are wrapped, (2) the attribute "legacyUsesWebservice : Boolean" which expresses a key difference in the behavior of wrappers, namely as per Belushi and Baghdadi (2007): if the access relationship between a web service and a legacy system in uni-lateral, i. e., a web service can access a legacy system but not vice versa, or if it is bilateral, i. e., the legacy applications and web service can access each other's functionality.

Service-orientation concepts

As stated in Sect. 1, a service is a "self-contained" component that provides a business functionality via a standardized interface (Papazoglou et al. 2008). Thus, in order to account for service-orientation and required non-functional and functional aspects (cf. R4–R6), e. g., new specializations of "Software" have been added. Being specializations of "Software", each of them offers some (business) functionality ("Function") and has some "Interface".

Amongst "Software" specializations, particularly the concept "Web service" needs to be discussed. Here we added attributes and a constraint that set a web service apart from software, in particular to express the reliance of web services on internet technologies (Gustavo et al. 2004, p. 124). We did so through the attributes "URI : String", stating the URI over which a web service is accessed, and a constraint on the interface definition language of the interface specified for the web service, stating that the interface definition language must be XML- or JSON-based.

To express QoS attributes of a web service or IT service, such as an average processing speed, we rely on the meta type software quality and its attributes. We do so because while these QoS attributes are relevant for a service specification, to the best of our knowledge there is nothing that conceptually sets QoS attributes *in general* apart from how we specify "regular" software qualities. This one can also see in the OMG's UML profile for specifying QoS (OMG 2008), which provides general concepts for specifying qualities, such as "QoSCharacteristic" denoting the quality one is interested in, or "QoSDimension" to express expected values, such as the expected minimum or maximum values.

Further, to express modularity, an important aspect of service-orientation (Papazoglou et al. 2008), we use the concept "Component", and additionally express its modularity through (1) the relationships "part_of" and "depends_on", corresponding to concept "Software", which express, respectively, how a component/software can be decomposed into smaller constituent components, and that there exist component dependencies; and (2) the constraint that a component should have at least one interface, which hides the internal workings of a software component, such as the programming language used. Note that we base our conceptualization of a component on the component-and-connector viewtype from Clements et al. (2002, p. 110), which concerns itself with the basic ideas of how, at run-time, several (software) subpieces together constitute a larger (software) system.

At this point it is important to note that we explicitly opted to introduce the general notion of "Component" to cover many similar ideas in service-orientation. Of particular note is the recently emerging term "microservices". Architectures organized around microservices follow principles similar to service-orientation (e. g., loose coupling, interfaces hiding internal complexity, Pautasso et al. 2017), yet place an additional emphasis on following particular principles. For example, for microservices centralized functionality, such as offered through an Enterprise Service Bus, is kept to a bare minimum (Koschmider 2017; Pautasso et al. 2017; Zimmermann 2017), and the software components acting as microservices typically execute fine-grained business capabilities. Yet, while for microservices SOA principles are followed more strictly, apart from the use of specific novel technologies (e. g., lightweight messaging systems cf. Zimmermann 2017) and management principles (prominently DevOps, to keep software management closely tied to software development cf. Zimmermann 2017), the underlying ideas remain – for now – similar to those of service-orientation and component-based/modular software development. Therefore, we opt to not introduce concepts related to microservices in their own right. However, since microservice architectures are an emerging and active research field, for future research our conceptualization of microservices may change.

Table 4: Examples of Concrete Syntax

Symbol	*Explanation*
	web service
	source code available
	source code unavailable

4.3 Concrete Syntax and Supporting Modeling Tool

After extending the abstract syntax of ITML, we have extended its concrete syntax with the aim to ensure its intuitiveness. To this end, we use well-established guidelines from Moody (2009) for designing cognitively effective visual notations (i. e., notations that are optimized for processing by the human mind). These guidelines include Semiotic Clarity, Perceptual Discriminability, Semantic Transparency, Visual Expressiveness and Graphic Economy. For example, semantic transparency implies that the meaning (semantics) of a symbol is clear (transparent) from its appearance alone (Moody 2009). In this case, the use of a suitable domain-specific graphical representation is considered important, as it allows domain stakeholders to quickly grasp the idea, thus, avoiding a long learning process. Tab. 4 shows exemplary symbols used to represent the language concepts of extended ITML.

The set of MEMO DSMLs (namely GoalML, OrgML-Organizational Structure and OrgML-Business Processes) has been implemented with the meta modeling software environment ADOxx (Fill and Karagiannis 2013). It has been made available under the name MEMO4ADO[2] (Bock and Frank 2016). However, due to the lack of support for intrinsic features and language level types in the ADOxx meta meta model (Fill and Karagiannis 2013)[3] , in order to implement MEMO modeling languages in ADOxx a redesign of the meta model was required, so that the desired domain aspects could all be modeled at exactly the same abstraction level (Bock and Frank 2016).

Being aware of this limitation, however, to benefit from the already implemented languages, we modify and extend the meta model of ITML within MEMO4ADO. The introduced extensions resulted also in new diagrams types that have been added. Thus, currently the tool offers the following diagram types: (1) IT Infrastructure Diagram, which focuses on modeling the overall IT landscape, including concepts related to service-orientation (such as a web service and its various attributes); (2) ITML Implementation Languages Diagram, which allows to model different implementation languages (and their characteristics) at the language level. The defined language level concepts are referenced within the IT Infrastructure Diagram type; (3) ITML Topic Diagram, which allows to define Functions and Data Topics. Similar to the ITML Implementation Languages diagram type, the modeled Topics are being referenced from the IT Infrastructure Diagram; (4) ITML File Exchange Format Diagram, which allows to model different types of file exchange that specify the format of the files being exchanged, which may be then referenced from the level of the main diagram; (5) ITML Library Diagram allows to define the used libraries, which again are referenced from the main diagram.

We summarize the used and implemented diagram types per SOA migration analysis phase in Tab. 5. Note that, as discussed in Sect. 3, the SOA migration analysis activities are not sequential. For example, “Legacy System Understanding” may partly overlap with “Candidate service

[2] The modeling tool can be downloaded from http://www.omilab.org/memo4ado

[3] The ADOxx meta meta model includes concepts to define meta classes, attributes, and relationships at level M_2, which can be instantiated at type level M_1 in the ADOxx Modeling Toolkit (Fill and Karagiannis 2013, 6–7). Yet, it does not support instantiating and managing instance populations at level M_0 that would represent instantiations of model elements from the level M_1.

identification and refinement", in the sense that functionalities are identified that one would like to offer as services. Also target system analysis may overlap with business context analysis, so that one takes into account the organization action system while designing the target system.

Table 5: MEMO SOA Migration Analysis Method: Phases and Corresponding Diagrams

Phase	*Exemplary Diagrams*
Business Context Analysis	Organizational Structure Diagram, Goal Diagram, Business Process Map
Legacy System Understanding	IT Infrastructure Diagram, ITML Library Diagram
Target System Understanding	IT Infrastructure Diagram
Candidate Service Identification and Refinement	IT Infrastructure Diagram, ITML Topic Diagram

5 Evaluation

We have conducted a threefold evaluation of the proposed approach: (1) against identified requirements, (2) by its application – using an extensively documented SOA migration scenario, we show how the extended ITML, in tandem with other languages from the MEMO language family, can be used to support a SOA migration analysis, (3) by showing the added value of MEMO with the extended ITML in a scenario-based comparison with ArchiMate, another language that is a promising candidate for SOA migration analysis. In the following, we briefly discuss the obtained results.

5.1 Requirements Fulfillment

Tab. 6 summarizes the fulfillment of the identified requirements by the extended ITML, when it is applied in tandem with other MEMO family languages. Thanks to the integration of the extended ITML with GoalML, OrgML (business processes) and OrgML (organizational structure) (e. g., via the relationship "supports") we have been able to account for the business concerns as required by R7. We have extended ITML to account for different types of services as well as for their non-functional aspects (R5 and R6). In turn, to support the analysis and decision which migration strategy should be followed (e. g., black-box or white-box), we have accounted for the detailed characteristics of legacy systems (R3) as well as accounted for concept allowing to express those (R1–R3, and R5). We also provided the extended version of the corresponding modeling tool (R9).

5.2 Applicability Evaluation: ACME Insurance – A SOA Migration Scenario

To evaluate the applicability as well as added value resulting out of the proposed extensions, we have applied the proposed method to scenarios documented in the literature, e. g., (Brandner et al. 2004; Rosen et al. 2008; Zimmermann et al. 2004). Here, we report on the most comprehensive one. Namely, we now model a running scenario of an ACME Insurance company discussed by Rosen et al. (2008, pp. 541–578). To this aim we use the extended version of ITML in tandem with the other MEMO languages, i. e., GoalML and OrgML. We follow the SOA analysis activities depicted in Fig. 1 as a guideline. After the extensive scenario, we briefly discuss the SOA migration scenario as documented by Brandner et al. (2004) and Zimmermann et al. (2004).

Business Context Analysis and Legacy System Understanding. ACME Insurance, a fictitious insurance company that is an aggregate of several SOA migration projects at actual insurance companies, specializes in two line-of-business: commercial auto and commercial property (Rosen et al. 2008, p. 541). ACME focuses on the full insurance life-cycle, which implies that it deals with preparing quotes and ratings, underwriting and servicing policies.

ACME Insurance decides to introduce service-orientation having the following key objectives in mind: (1) to share IT capabilities across different Lines of Business, (2) to achieve synergy between

Table 6: Requirements Fulfillment – Summary

R	*Requirement Description*	*Explanation*
1	Expressing IT landscape elements	The basic elements of IT landscape have been accounted for, e. g., Database, Database management system, Middleware, Server. To allow for expressing additional concepts, not revealed by the conducted analysis of migration related concepts, the concept Software may be used.
2	Expressing the dependencies between IT landscape elements	We have accounted for a rich set of domain specific relationships, e. g., uses, provides, runs on, based on.
3	Characteristics of legacy systems and dependencies	A set of attributes/relationships has been added to facilitate the analysis of legacy systems, e. g., mission criticality, source code availability, implementation language, code complexity, availability of documentation, strength of support.
4	Service and its types	A set of Software specializations (e. g., IT Service, Web Service, Component) has been added (all with additional attributes and constraints) in order to express different aspects of service-orientation
5	Relating a service to its underlying implementation.	Relevant relationships have been defined: wraps, provides, runs on, uses
6	Quality attributes of services	Quality of Service attributes are specified using the meta type software quality and its attributes.
7	Dependencies between the IT landscape and the organization action system	Different relationships and association types (e. g., SpecificSupport, UseContext) allow to link IT infrastructure elements with business processes;
8	Semantically rich concepts	Each extended ITML concept, including relationships to concepts from other MEMO languages, has been equipped with a rich set of domain-specific attributes.
9	Modeling Tool	The implementation of ITML in the MEMO4ADO tool has been modified to account for the introduced changes. Also the additional integration with the concepts from other MEMO languages has been accounted for.

legacy applications, (3) to address the fragmentation of the IT landscape, and (4) to ease addition of new insurance products.

For scoping reasons, we focus our case study on the process of insurance underwriting, which is concerned with matching a customer profile to an appropriate insurance package and fee. The following are key process steps for underwriting: gathering information for quote, underwriting, and sending a quote to the customer.

Fig. 4 shows the business context in terms of this underwriting process, and how it is supported by the current IT infrastructure. We focus our discussion on outstanding features of the *extended* ITML and MEMO for the SOA migration analysis, marked by labels (1) – (3).

First, the *extended* ITML, in conjunction with OrgML, helps identifying how current business processes are supported by IT infrastructure (in line with R7, Tab. 1). This provides a baseline IT infrastructure on which the SOA should build, and helps with identifying key IT functionality that is important to transform into service-orientation.

In Fig. 4, this feature is illustrated by labels (1a) and (1b). These mark the "supports" relationship between the functions that are part of the function "Car policy administration" as provided by the application "Car policy and products administration", namely: the function "Establish policy submission" supporting the business processes "Gather information for quote", and the function "Rate insurance policy" which supports the business process "Underwrite quote". Note that we show the functions of the car policy application only, to avoid overcrowding the model. Thus, the extended ITML helps us identify the legacy system functionality and its support to the organizational action system by (a) using functions to

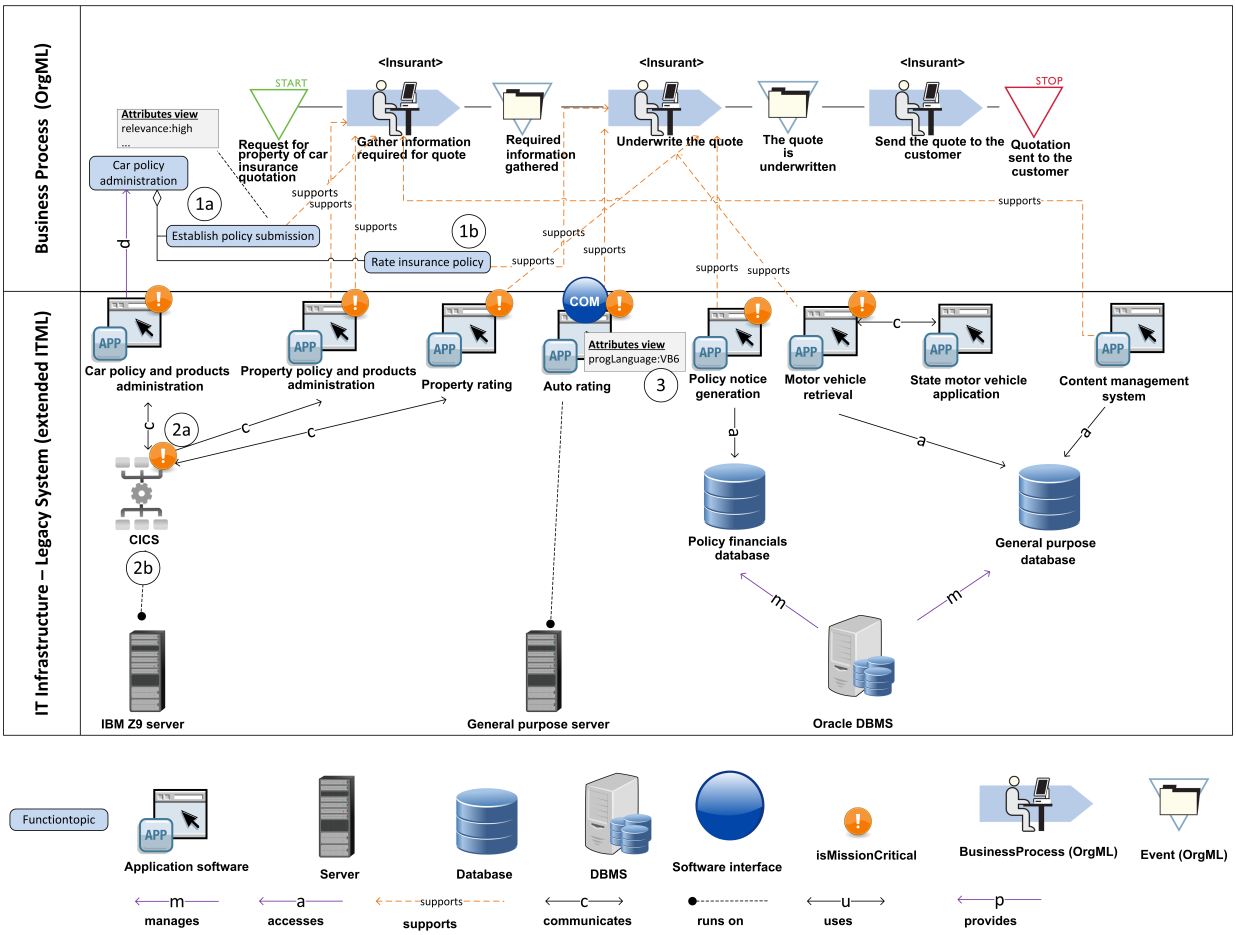

Figure 4: ACME Insurance's current system and its business context, modeled in the extended ITML and OrgML

express the functionality provided by a software, including potentially finegrained functionalities of which an abstract functionality is composed. For example: we can express that the function "Car policy administration" is composed of functions "Establish policy submission" and "Rate insurance policy"; (b) annotating the "supports" relationship with an importance score (e. g., in this case: "relevance = high"); (c) considering all business processes that are supported by the functionality, as provided by a particular application. Note that we can further mark important applications with the attribute "isMissionCritical : Boolean".

Second, the *extended* ITML helps identify specific relationships between IT infrastructure elements, in line with R2 (Tab. 1). In Fig. 4 this is illustrated by labels (2a) and (2b). The label located on the "communicates" relationships expresses that three applications communicate with CICS middleware[4] . In turn, label (2b) on the "runs on" relationship indicates that the CICS middleware runs on the Z9 server, but not the other way around.

[4] CICS middleware supports the processing of large amounts of transactions, and often runs on (Z-series) mainframes. Despite it being costly to maintain, CICS is still popular in organizations that deal with large volumes of transactions, being mostly banks and insurance companies.

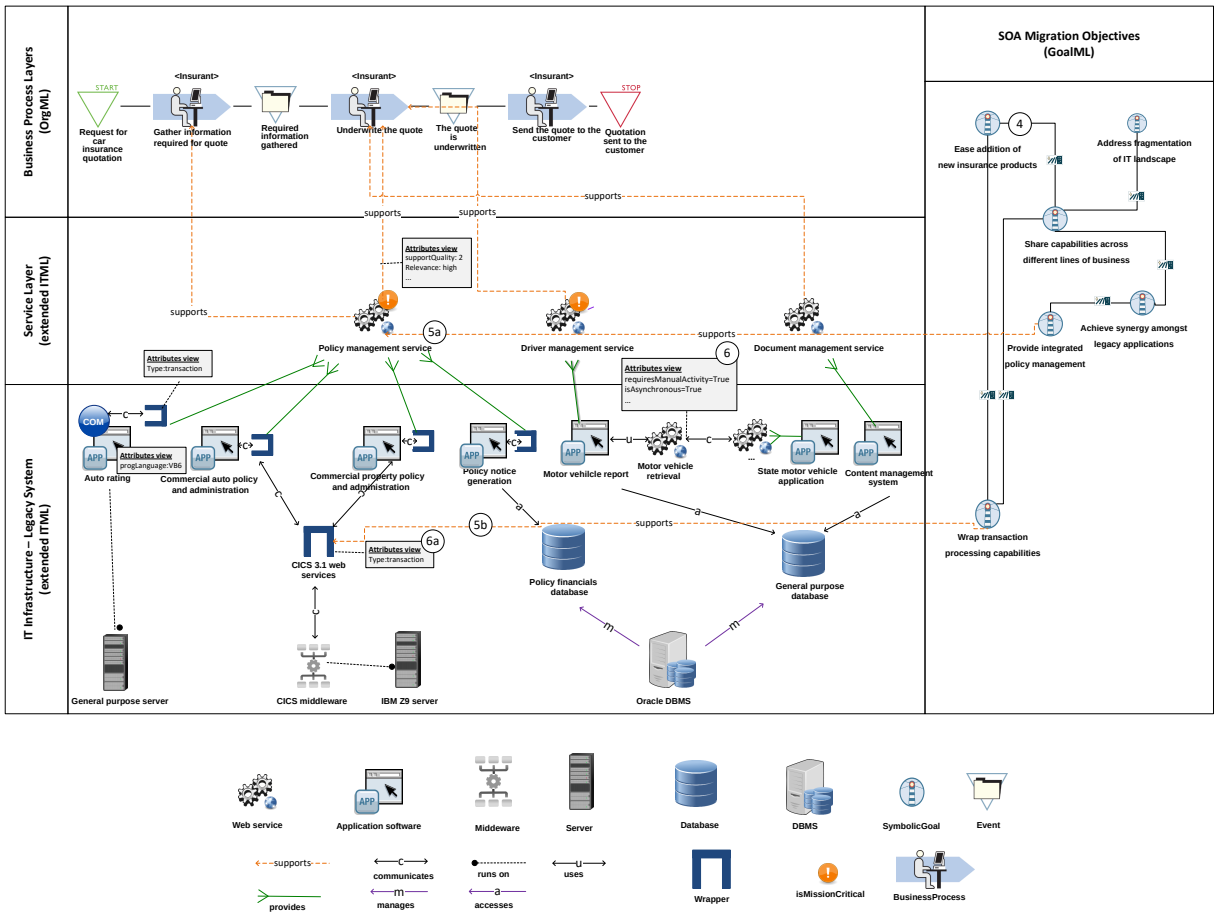

Figure 5: ACME Insurance's desired service orientation, modeled in the extended ITML, OrgML and GoalML

Third, the *extended ITML* can help describe the non-functional attributes of legacy systems, which in turn helps understanding the legacy system. In Fig. 4 this is exemplified by label (3), which allows an analyst to understand why the ACME Insurance's "Auto rating" application has a COM interface[5] , namely: because it is one of few applications that has been programmed in Visual Basic 6, and thus, needs an interface for applications programmed in different languages to be able to interact with it.

[5] We do not want to suggest that COM component technology is a superficial "shell". We are aware that (D)COM consists of an extensive set of supporting libraries, and that it is based on the wider notion of component-based development. However, it is the interface only that we focus on here, since this determines how one interacts with the application.

Candidate Service Identification and Refinement and Target System Understanding. Fig. 5 shows the ACME's to-be service-oriented IT infrastructure modeled in the extended version of ITML. In it, we find the following candidate services (in line with Rosen et al. 2008, p. 548): Policy management service, Driver management service, and Document management service. Besides reusing current IT capabilities that play an important part in the current IT infrastructure (as identified in Fig. 4), the *goal analysis* by means of GoalML plays an important role in identifying these services.

Labels (4), (5a), and (5b) illustrate the role that the goal analysis plays in service identification. First, the goal analysis can be used to structure the plain text list of SOA objectives identified

(Rosen et al. 2008, p. 542), and to translate high level goals into more concrete ones. For example, in terms of structuring, we see that fulfilling the SOA migration objective "Share capabilities across different lines of business" contributes to fulfilling the objectives "Ease addition of new insurance products", and "Address fragmentation of IT landscape". Furthermore, we find that the objective "Achieve synergy amongst legacy applications" positively contributes to the objective "Share capabilities across different lines of business". So, instead of objectives that are at first sight orthogonal to each other, we can see how they are interrelated, which helps identify how fulfilling one goal can contribute to (or negate) the fulfillment of another goal. Also, we see how the goal analysis can be used to translate high-level goals into more concrete ones. For example, the objective "Provide integrated policy management" is introduced as a concrete goal to achieve "Achieve synergy amongst legacy applications", whereas "Wrap transaction processing capabilities" is introduced to "Ease addition of new insurance products" (the idea being that newly build non-mainframe applications can, thus, capitalize more easily on transaction processing capabilities) as well as to "Share capabilities across different lines of business".

Labels (5a) and (5b) show how the identified objectives translate into service-orientation. By means of the association class "supports" we identify how (1) a "Policy management service" is introduced as a response to the SOA migration objective "Provide integrated policy management", and (2) the wrapper CICS 3.1 Web service is introduced to allow web services access to mainframe transaction processing capabilities.

Finally, labels (6a) and (6b) illustrate further the ability of expressing non-functional attributes in the *extended* ITML. Particularly, these labels illustrate how we can use attributes to express asynchronous communication between the web service "State Motor Vehicle Reporting" and the web service "Motor Vehicle Report" from ACME Insurance. Furthermore, as can be seen from Fig. 5, attributes allow us to express that the CICS 3.1 web services wrapper is of type "transaction", thus, allowing for a bi-directional communication between different software (as mentioned in Sect. 3 in case of a session-based wrapper, the communication relationship would have to be unidirectional).

Short summary of findings of a different SOA migration scenario

We also confronted extended ITML to a SOA migration scenario from the financial industry, as documented by Brandner et al. (2004) and Zimmermann et al. (2004). The case concerns the development of a standardized interface for a shared service centre providing banking services. On the one hand, this case follows a largely technical narrative, making it hard to test business-oriented aspects of extended ITML. On the other hand, the cases showed that extended ITML is capable of expressing the most salient technical decisions, being amongst others: (1) the decision to use WSDL as the only interface description language for the introduced web services, through the extended ITML concepts "WebService" and "Interface", and the Interface attribute descriptionLanguage; (2) the reliance on different middlewares (CICS, and Websphere) for the connection to the back-end. However, unfortunately, Brandner et al. (2004) and Zimmermann et al. (2004) did not provide details of the back end legacy systems, hence we could not fully test the expressiveness of extended ITML with regards to connecting heterogeneous system elements for this case.

5.3 Comparison with ArchiMate

In the previous section, we have illustrated how an EM approach with a focus on IT infrastructure modeling can play a meaningful role in the SOA migration analysis. In line with our requirements, it can be used to: (1) inventory relevant IT infrastructure assets and their relationships, such as the desire to keep using legacy bulk transaction processing functionality via a wrapper, (2) express non-functional attributes, such as the necessity of a web service to support asynchronous communication, as well as a wrapper supporting bi-directional

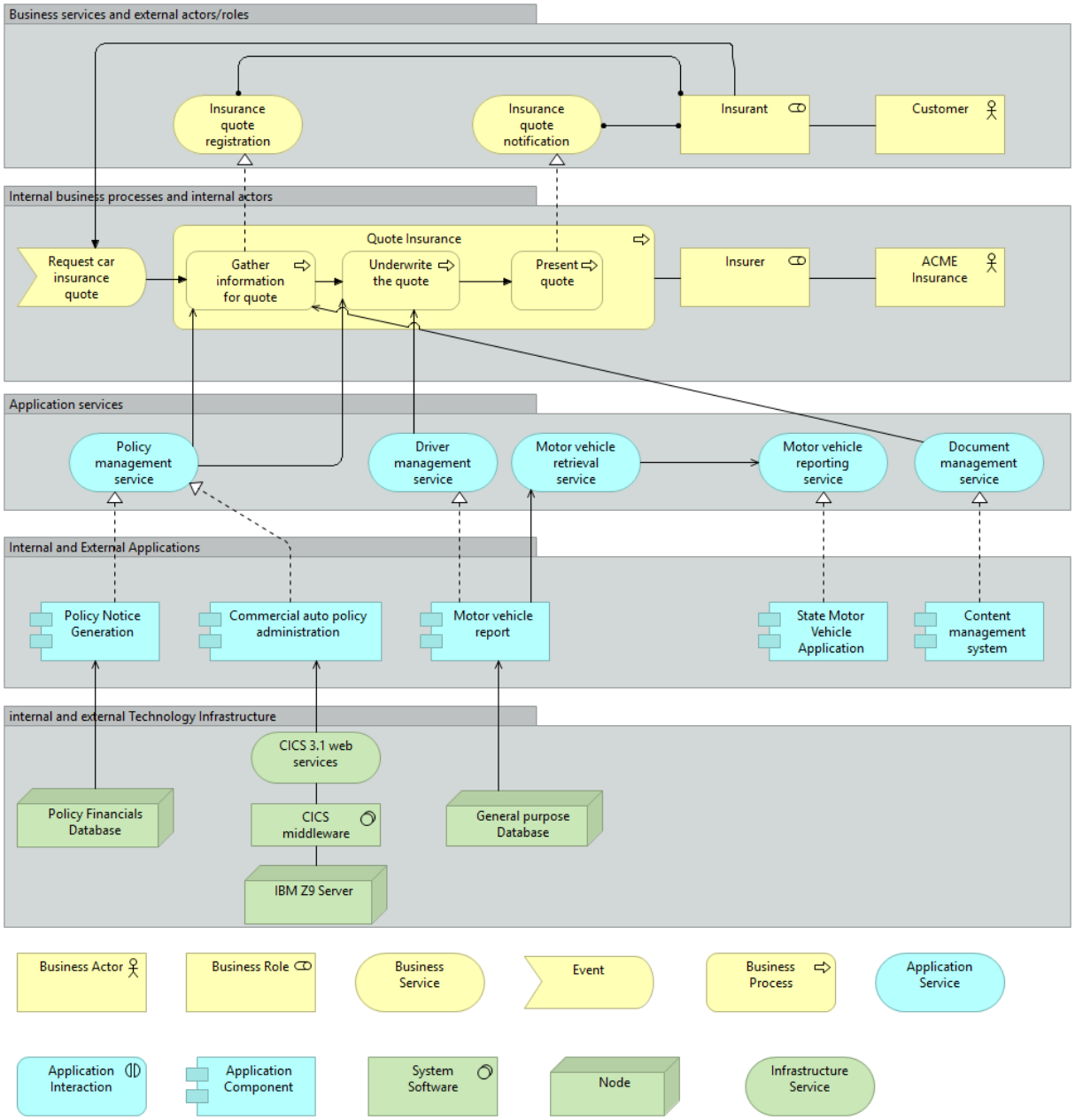

Figure 6: ACME Insurance's desired service orientation, modeled in ArchiMate

communication, and (3) an overview of how IT functionality supports business processes, such as the web services "Policy management service" and "Driver management service" both supporting the business process "Underwrite the quote".

To illustrate the added expressiveness of the *extended* ITML, let us briefly contrast it with ArchiMate, another promising candidate identified in Sect 3.3 for modeling service-orientation. The analysis of Fig. 6 (depicting the to-be service-orientation of ACME Insurance) shows that ArchiMate lacks expressiveness for IT infrastructure concepts that one would need for a meaningful SOA migration analysis. This particularly con-

cerns ArchiMate's multi-interpretable concepts and the lack of domain-specific attributes.

Multi-interpretable concepts. ArchiMate is on purpose designed with a limited set of concepts in mind, while at the same time maintaining sufficient expressiveness for most enterprise architecture modeling tasks (Lankhorst et al. 2010). This has resulted in a set of abstract concepts that, while largely suitable for the enterprise architecture domain, is too multi-interpretable for meaningful *domain-specific* analyses. Often, when it comes to detailed domain-specific analyses, the abstract syntax offered by ArchiMate allows one to create relationships between concepts that are illegal from a domain point of view.

Consider the "Server" and "Middleware" concepts from the IT infrastructure domain. As depicted in Fig. 6, in the ACME Insurance scenario, but more generally also for large insurance and banking organizations, ACME Insurance runs CICS middleware on its Z9 server. While in the to-be situation CICS is being offered via a wrapper (CICS 3.1 web services), ACME Insurance considers it important to maintain this transaction processing capability.

Whereas one could in principle express this server-middleware relationship in ArchiMate, due to its multi-interpretable nature, one might as well express it the other way around: that the Z9 server runs on the middleware[6] . However, from the point of view of the IT infrastructure domain, this is illogical.

Domain-specific attributes. In addition to having multi-interpretable concepts that can be related in different (sometimes illegal) ways, ArchiMate lacks domain-specific attributes. Consider attributes specific to the "WebService" concept. In the ACME Insurance scenario we need to especially express attributes for web services with regards to using either synchronous or asynchronous communication. To ACME Insurance, expressing synchronicity of communication is relevant for retrieving the Motor Vehicle report of a (potential) customer. If ACME Insurance already has the full data locally, it can simply retrieve this report from a local database with a synchronous communication model. However, if insufficient data is available, ACME Insurance has to request this information from the state motor vehicle authorities. This request often leads to a delayed response, thus, requiring the web service to be implemented taking into consideration the asynchronous communication model. ArchiMate's abstract syntax lacks such attributes specific to IT infrastructures.

Of course, the decision to include attributes into a language depends on the analysis scenario to be supported – in this case what needs to be in place to enable service-orientation for ACME Insurance. However, the discussed examples have shown that dedicated analyses for the IT infrastructure domain are insufficiently enabled by using ArchiMate as such.

Thus, as shown in Sect. 5.2, dedicated support from the *extended* ITML allows us to express SOA migration concerns that are insufficiently expressed by ArchiMate. Particularly, while ArchiMate allows for linking business and IT concerns, as we have shown it lacks support in both modeling dedicated IT infrastructure concepts and their relationships, and in expressing non-functional aspects.

6 Discussion and Conclusions

In this paper, based upon a literature review on SOA migration analysis, and characteristics of migration projects, we have identified a set of requirements that conceptual modeling languages should fulfill in order to support SOA migration. Subsequently, a survey of existing modeling approaches has shown that none of them fulfills all

[6] The ArchiMate concepts closest to "Server" and "Middleware" would be the "Node" concept or its specialization "System software", which are defined respectively as "... a computational resource upon which artifacts may be stored or deployed for execution" (Node, cf. The Open Group 2013) and as "... a software environment for specific types of components and objects that are deployed on it in the form of artifacts" (Systems software, cf. The Open Group 2013). However, by expressing both concepts as nodes in ArchiMate (or specializations thereof) one might as well let a middleware run a server.

requirements, but that one approach comes closest: MEMO. Following this, we have extended MEMO, in particular its IT infrastructure modeling language ITML. Then, we have shown how the *extended* ITML used together with other MEMO languages can be applied to a realistic SOA migration scenario. Finally, we discussed software tool support, and showed the added value of the *extended* ITML by contrasting it with the same migration scenario modeled in ArchiMate, another candidate modeling language coming close to satisfying our requirements. Here we have shown that the *extended* ITML offers an added value compared to ArchiMate in terms of expressing domain-specific attributes, as well as concepts and relationships specific to IT infrastructure (such as a "runs on" relationship with specific meaning).

While extending ITML, in line with our purposes we have focused on concepts required for a detailed analysis of IT infrastructure. Mainly we did this in terms of legacy systems characteristics and characteristics for expressing service-orientation.

Regarding legacy systems, an important concern is that specifications of legacy systems rarely exist and that hardly anything is documented (cf. Khadka et al. 2013b; Lewis et al. 2006). So it follows that extending the scope of a modeling language is not enough as still the way to acquire the required information should be accounted for. Therefore, the next step is to check how the existing tools and methods aiming at analysis and re-engineering of legacy systems discussed in Sect. 2.1, such as the various feature analysis approaches, can be used to acquire information relevant for populating the models. Furthermore, we provided basic concepts for expressing databases, and how they can be related to service-orientation. However, while sufficient for our scenario, integration of data from databases into a SOA is also a more complex research issue. For example, one should be careful with the database-to-schema conversion discussed by Rosen et al. (2008, p. 199). Here it is pointed out that simply converting a database to XML results in a tight-coupling between the original database and its XML representation. This tight-coupling can lead to misunderstandings amongst stakeholders not directly familiar with the database's schema, and worse still: it can lead to difficulties when the database is changed, since the change somehow needs to be reflected into the XML messages. So, in database migration, one should take care of structurally migrating data structures, an area where modeling can be particularly helpful.

Regarding service-orientation, the *extended* ITML provides a basic set of concepts required to express many of the concerns relevant to service-orientation. These include a link between business concerns and IT concerns, a link between a service and the underlying IT infrastructure concepts, and service attributes. Although we have expressed the non-functional aspects of (web) services pragmatically, we are well aware, as also witnessed by, among others D'Ambrogio (2006) and Jureta et al. (2009), that expressing such non-functional attributes in a conceptual model is more complex than simply adding a meta type "Software Quality" (as we did while extending ITML). For one, depending on the purpose of the model, one has to consider aggregate versus detailed service qualities, such as reliability being specified into an average number of failures over a certain time period, and the average duration of a failure (D'Ambrogio 2006). In addition, one may have to express contractual agreements relating to QoS, as well as the possibility to add (relative) importance values (Jureta et al. 2009).

Regarding the language architecture used, although the application of the MEMO meta modeling language (MML) allowed us to take advantage of the intrinsic features and relationships, and thus, to refer to the instance level, we faced modeling challenges due to to the restrictions imposed by the type/instance dichotomy and the semantic differences between instantiation and specialization (cf. Frank 2014). The MML language architecture as well as other languages following a "traditional" modeling paradigm (i. e., in line with the Meta Object Facility language architecture (OMG 2015b)) offer no satisfying solution to the above-mentioned challenges (cf. Atkinson and Kühne 2001, 2008;

Frank 2014; Kaczmarek-Heß and Kinderen 2017). As a result, in the proposed meta model we needed to restrict the meta types to a few generic ones, in order to find a balance between productivity and reuse, in line with the identified requirements. Nevertheless, taking into account the limitations imposed by the meta modeling architecture, as a next step of our research we undertake an attempt to apply a multilevel modeling language like, e. g., the Flexible Meta Modeling and Execution Language, FMML^{x} (Frank 2014). Such a multilevel modeling allows to consider an arbitrary number of modeling levels, instead of the two that we currently rely on.

Finally, the evaluation of the performed extensions is limited in the sense of using a documented case study. Although the documented case study is realistic and detailed, it would be worthwhile to gain additional insights from application to a real world case. For one, we (implicitly) assume that the concepts from the different modeling languages are understood by domain experts (IT analysts, process managers, etc.) and can be used to communicate with them. However, this assumption should be verified.

References

Almonaies A. A., Cordy J. R., Dean T. R. (2010) Legacy system evolution towards service-oriented architecture. In: International Workshop on SOA Migration and Evolution, SOAME 2010. IEEE Computer Society, Washington, DC, USA, pp. 53–62

Alwadain A., Fielt E., Korthaus A., Rosemann M. (2016) Empirical Insights into the Development of a Service-oriented Enterprise Architecture. In: Data Knowledge Engineering 105(C), pp. 39–52

Arsanjani A., Ghosh S., Allam A., Abdollah T., Ganapathy S., Holley K. (2008) SOMA: A method for developing service-oriented solutions. In: IBM System Journal 47(3), pp. 377–396

Atkinson C., Kühne T. (2001) The Essence of Multilevel Metamodeling. In: Gogolla M., Kobryn C. (eds.) Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools. Springer, London, UK, pp. 19–33

Atkinson C., Kühne T. (2008) Reducing accidental complexity in domain models. In: Software & Systems Modeling 7(3), pp. 345–359

Baghdadi Y., Al-Bulushi W. (2015) A Guidance Process to Modernize Legacy Applications for SOA. In: Service Oriented Computing and Applications 9(1), pp. 41–58

Balasubramaniam S., Lewis G. A., Morris E., Simanta S., Smith D. (2008) SMART: Application of a Method for Migration of Legacy Systems to SOA Environments. In: Bouguettaya A., Krueger I., Margaria T. (eds.) Service-Oriented Computing, ICSOC 2008. Springer, pp. 678–690

Bell M. (2008) Service-oriented modeling. Wiley Publishing

Belushi W. A., Baghdadi Y. (2007) An Approach to Wrap Legacy Applications into Web Services. In: Intenational Conference on Service Systems and Service Management. IEEE Computer Society, Washington, DC, USA, pp. 1–6

Bhallamudi P., Tilley S. (2011) SOA migration case studies and lessons learned. In: 2011 IEEE International Systems Conference. IEEE Computer Society, Washington, DC, USA, pp. 123–128

Bisbal J., Lawless D., Wu B., Grimson J. (1999) Legacy Information Systems: Issues and Directions. In: IEEE Software 16(5), pp. 103–111

Bock A., Frank U. (2016) Multi-perspective Enterprise Modeling – Conceptual Foundation and Implementation with ADO*xx*. In: Karagiannis D., Mayr H. C., Mylopoulos J. (eds.) Domain-Specific Conceptual Modeling, Concepts, Methods and Tools. Springer, Cham, pp. 241–267

Bock A., Kaczmarek M., Overbeek S., Heß M. (2014) A Comparative Analysis of Selected Enterprise Modeling Approaches. In: Frank U., Loucopoulos P., Pastor Ó., Petrounias I. (eds.) The Practice of Enterprise Modeling. Springer, pp. 148–163

Bouguettaya A., Singh M., Huhns M., Sheng Q. Z., Dong H., Yu Q., Neiat A. G., Mistry S., Benatallah B., Medjahed B., Ouzzani M., Casati F., Liu X., Wang H., Georgakopoulos D., Chen L., Nepal S., Malik Z., Erradi A., Wang Y., Blake B., Dustdar S., Leymann F., Papazoglou M. (2017) A Service Computing Manifesto: The Next 10 Years. In: Communications of the ACM 60(4), pp. 64–72

Brandner M., Craes M., Oellermann F., Zimmermann O. (2004) Web services-oriented architecture in production in the finance industry. In: Informatik-Spektrum 27(2), pp. 136–145

Bresciani P., Perini A., Giorgini P., Giunchiglia F., Mylopoulos J. (2004) Tropos: An Agent-Oriented Software Development Methodology. In: Autonomous Agents and Multi-Agent Systems 8(3), pp. 203–236

Brogi A., Soldani J., Wang P. (2014) TOSCA in a Nutshell: Promises and Perspectives. In: Villari M., Zimmermann W., Lau K.-K. (eds.) Service-Oriented and Cloud Computing: Third European Conference, ESOCC 2014, Manchester, UK, September 2-4, 2014. Springer, pp. 171–186

Bucher T., Fischer R., Kurpjuweit S., Winter R. (2006) Analysis and Application Scenarios of Enterprise Architecture: An Exploratory Study. In: 10th IEEE International Enterprise Distributed Object Computing Conference Workshops. IEEE Computer Society, Washington, DC, USA, p. 28

Clements P., Garlan D., Bass L., Stafford J., Nord R., Ivers J., Little R. (2002) Documenting software architectures: views and beyond. Pearson Education

Comella-Dorda S., Wallnau K., Seacord R., Robert J. (2000) A Survey of Legacy System Modernization Approaches. CMU/SEI-2000-TN-003. Software Engineering Institute, Carnegie Mellon University. Pittsburgh, PA

D'Ambrogio A. (2006) A Model-driven WSDL Extension for Describing the QoS of Web Services. In: IEEE International Conference on Web Services. ICWS 2006. IEEE Computer Society, Washington, DC, USA, pp. 789–796

Dardenne A., van Lamsweerde A., Fickas S. (1993) Goal-directed Requirements Acquisition. In: Science of Computer Programming 20(1-2), pp. 3–50

de Kinderen S., Kaczmarek-Heß M. (2017) Enterprise Modeling Support for SOA Migration. In: Leimeister J., Brenner W. (eds.) Proceedings der 13. Internationalen Tagung Wirtschaftsinformatik, WI 2017. AIS

Deiters C., Rausch A., Schindler M. (2013) Using spectral clustering to automate identification and optimization of component structures. In: 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE 2013, San Francisco, CA, USA, May 25-26, 2013. IEEE Computer Society, Washington, DC, USA, pp. 14–20

Dietz J. L. G. (2006) Enterprise Ontology: Theory and Methodology. Springer

Ducasse S., Lanza M., Tichelaar S. (2000) MOOSE: an Extensible Language-Independent Environment for Reengineering Object-Oriented Systems. In: Gray J., Scott L., Ferguson I. (eds.) The Second International Symposium on Constructing Software Engineering Tools – Workshop Session. ACM, New York, NY, USA, pp. 24–30

Engelsman W., Wieringa R. (2014) Understandability of Goal-Oriented Requirements Engineering Concepts for Enterprise Architects. In: Jarke M., Mylopoulos J., Quix C., Rolland C., Manolopoulos Y., Mouratidis H., Horkoff J. (eds.) Advanced Information Systems Engineering. Springer, Cham, pp. 105–119

Fill H., Karagiannis D. (2013) On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform. In: Enterprise Modelling and Information Systems Architectures 8(1), pp. 4–25

Frank U. (1994) Multiperspektivische Unternehmensmodellierung. Theoretischer Hintergrund und Entwurf einer objektorientierten Entwicklungsumgebung. Oldenbourg, München

Frank U. (2010) Outline of a Method for Designing Domain-Specific Modelling Languages. ICB Research Report 42. University of Duisburg-Essen. Essen

Frank U. (2011) The MEMO Meta modeling Language (MML) and Language Architecture. 2nd Edition. ICB-Research Report 43. University of Duisburg-Essen. Essen

Frank U. (2012) Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. In: Software and Systems Modeling 13(3), pp. 941–962

Frank U. (2013) Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines. In: Reinhartz-Berger I., Sturm A., Clark T., Cohen S., Bettin J. (eds.) Domain Engineering: Product Lines, Languages, and Conceptual Models. Springer, pp. 133–157

Frank U. (2014) Multilevel Modeling – Toward a New Paradigm of Conceptual Modeling and Information Systems Design. In: Business & Information Systems Engineering 6(6), pp. 319–337

Fuhr A., Horn T., Riediger V., Winter A. (2013) Model-driven software migration into service-oriented architectures. In: Computer Science-Research and Development 28(1), pp. 65–84

Goldstein A., Frank U. (2016) Components of a multi-perspective modeling method for designing and managing IT security systems. In: Information Systems and e-Business Management 14(1), pp. 101–140

Gustavo A., Casati F., Kuno H., Machiraju V. (2004) Web services: concepts, architectures and applications. Springer

Hanschke I. (2010) IT Landscape Management In: Strategic IT Management: A Toolkit for Enterprise Architecture Management. Springer, pp. 105–217

Heise D. (2013) Unternehmensmodell-basiertes IT-Kostenmanagement als Bestandteil eines integrativen IT-Controllings. Logos, Berlin

Hirschheim R., Welke R. J., Schwarz A. (2010) Service-Oriented Architecture: Myths, Realities, and a Maturity Model. In: MIS Quarterly Executive 9(1), pp. 37–48

Jamshidi P., Ahmad A., Pahl C. (2013) Cloud Migration Research: A Systematic Review. In: IEEE Transactions on Cloud Computing 1(2), pp. 142–157

Jureta I. J., Herssens C., Faulkner S. (2009) A comprehensive quality model for service-oriented systems. In: Software Quality Journal 17(1), pp. 65–98

Kaczmarek-Heß M., de Kinderen S. (2017) A Multilevel Model of IT Platforms for the Needs of Enterprise IT Landscape Analyses. In: Business & Information Systems Engineering 59(9), pp. 315–329

Khadka R., Batlajery B. V., Saeidi A. M., Jansen S., Hage J. (2014) How Do Professionals Perceive Legacy Systems and Software Modernization? In: Proceedings of the 36th International Conference on Software Engineering. ICSE 2014. ACM, Hyderabad, India, pp. 36–47

Khadka R., Reijnders G., Saeidi A., Jansen S., Hage J. (2011) A method engineering based legacy to SOA migration method. In: 27th IEEE International Conference on Software Maintenance, ICSM 2011. IEEE Computer Society, Los Alamitos, CA, USA, pp. 163–172

Khadka R., Saeidi A., Idu A., Hage J., Jansen S. (2013a) Legacy to SOA evolution: A systematic literature review. In: Ionita A. D., Litoiu M., Lewis G. (eds.) Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments. IGI Global, Hershey, PA, pp. 40–70

Khadka R., Saeidi A., Jansen S., Hage J., Haas G. P. (2013b) Migrating a large scale legacy application to SOA: Challenges and lessons learned. In: 20th Working Conference on Reverse Engineering, WCRE 2013. IEEE Computer Society, Los Alamitos, CA, USA, pp. 425–432

Kirchner L. (2008) Eine Methode zur Unterstützung des IT-Managements im Rahmen der Unternehmensmodellierung. Logos, Berlin

Koschmider A. (2017) Microservices-based Business Process Model Execution. In: 8th International Workshop on Enterprise Modeling and Information Systems Architectures. CEUS WS

Lankhorst M. (2013) Enterprise Architecture at Work: modeling, Communication and Analysis, 3rd ed. The Enterprise Engineering Series. Springer

Lankhorst M. M., Proper H. A., Jonkers H. (2010) The anatomy of the ArchiMate language. In: International Journal of Information System Modeling and Design 1(1), pp. 1–32

Lewis G., Morris E., Smith D. (2006) Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In: Conference on Software Maintenance and Reengineering. IEEE Computer Society, Washington, DC, USA, pp. 9–23

Lewis G., Morris E., Smith D., O'Brien L. (2005) Service-Oriented Migration and Reuse Technique (SMART). In: 13th IEEE International Workshop on Software Technology and Engineering Practice. STEP 2005. IEEE Computer Society, Washington, DC, USA, pp. 222–229

Lwakatare L. E., Kuvaja P., Oivo M. (2015) Dimensions of DevOps. In: Lassenius C., Dingsøyr T., Paasivaara M. (eds.) Agile Processes in Software Engineering and Extreme Programming: 16th International Conference, XP 2015, Helsinki, Finland, May 25-29, 2015. Springer, Cham, pp. 212–217

MacLennan E., Van Belle J.-P. (2014) Factors affecting the organizational adoption of service-oriented architecture (SOA). In: Information Systems and e-Business Management 12(1), pp. 71–100

Millham R. (2010) Migration of a Legacy Procedural System to Service-Oriented Computing Using Feature Analysis. In: CISIS 2010, The Fourth International Conference on Complex, Intelligent and Software Intensive Systems, Krakow, Poland, 15-18 February 2010. IEEE Computer Society, Washington, DC, USA, pp. 538–543

Moody D. L. (2009) The Physics of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. In: IEEE Transactions on Software Engineering 35(6), pp. 756–779

OASIS (2013) Topology and Orchestration Specification for Cloud Applications Version 1.0. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf

OMG (2008) Profile for modeling quality of service and fault tolerance characteristics and mechanisms, version 1.1. http://www.omg.org/spec/QFTP/About-QFTP/

OMG (2011) Business Process Model and Notation (BPMN), Version 2.0. http://www.omg.org/spec/BPMN/2.0

OMG (2012) Service oriented architecture Modeling Language (SoaML), version 1.0.1. http://www.omg.org/spec/SoaML/About-SoaML/

OMG (2015a) The Business Motivation Model (BMM), version 1.3. http://www.omg.org/spec/BMM/1.3/

OMG (2015b) The Meta Object Facility, Version 1.4. http://www.omg.org/spec/MOF/1.4/About-MOF/

OMG (2015c) The OMG Unified Modeling Language (OMG UML), version 2.5. http://www.omg.org/spec/UML/2.5/

Österle H., Becker J., Frank U., Hess T., Karagiannis D., Krcmar H., Loos P., Mertens P., Oberweis A., Sinz E. J. (2010) Memorandum zur gestaltungsorientierten Wirtschaftsinformatik. In: Zeitschrift fuer betriebswirtschaftliche Forschung 62(6), pp. 664–672

Overbeek S., Frank U., Köhling C. (2015) A language for multi-perspective goal modelling: Challenges, quirements and solutions. In: Computer Standards & Interfaces 38, pp. 1–16

Papazoglou M. P., van den Heuvel W.-J. (2006) Service-oriented design and development methodology. In: International Journal of Web Engineering and Technology 2(4), pp. 412–442

Papazoglou M. P., Traverso P., Dustdar S., Leymann F. (2008) Service-oriented computing: a research roadmap. In: International Journal of Computational Intelligence Systems 17(02), pp. 223–255

Pautasso C., Zimmermann O., Amundsen M., Lewis J., Josuttis N. (2017) Microservices in Practice, Part 1: Reality Check and Service Design. In: IEEE Software 34(1), pp. 91–98

Rabelo R. J., Noran O., Bernus P. (2015) Towards the Next Generation Service Oriented Enterprise Architecture. In: IEEE 19th International Enterprise Distributed Object Computing Workshop. IEEE Computer Society, Los Alamitos, CA, USA, pp. 91–100

Razavian M., Gordijn J. (2015) Consonance Between Economic and IT Services: Finding the Balance Between Conflicting Requirements. In: Fricker S. A., Schneider K. (eds.) Requirements Engineering: Foundation for Software Quality. Springer, Cham, pp. 148–163

Razavian M., Lago P. (2010) Towards a Conceptual Framework for Legacy to SOA Migration. In: Dan A., Gittler F., Toumani F. (eds.) Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops. Springer, pp. 445–455

Razavian M., Lago P. (2015) A systematic literature review on SOA migration. In: Journal of Software: Evolution and Process 27(5), pp. 337–372

Rosen M., Lublinsky B., Smith K. T., Balcer M. J. (2008) Applied SOA: Service-Oriented Architecture and Design Strategies. Wiley Publishing

Salama R., Aly S. (2008) A Decision Making Tool for the Selection of Service Oriented-Based Legacy Systems Modernization Strategies. In: Arabnia H. R., Reza H. (eds.) Proceedings of the International Conference on Software Engineering Research & Practice, July 14-17, 2008, Las Vegas Nevada, USA, 2 Volumes. CSREA Press, pp. 396–402

Sandkuhl K., Wißotzki M., Stirna J. (2013) Unternehmensmodellierung: Grundlagen, Methode und Praktiken. Springer Vieweg

Scheer A.-W. (2001) ARIS – Modellierungsmethoden, Metamodelle, Anwendungen, 4th ed. Springer

Silva F. G., de Menezes J. S., Lima J. d. S., França J. M., do Nascimento R. P., Soares M. S. (2015) An Experience of using SoaML for Modeling a Service-Oriented Architecture for Health Information Systems. In: 17th International Conference on Enterprise Information Systems - Volume 3. ICEIS 2015. SCITEPRESS - Science and Technology Publications, Lda, Barcelona, Spain, pp. 322–327

Sneed H., Heilmann H., Wolf E. (2016) Softwaremigration in der Praxis: Übertragung alter Softwaresysteme in eine moderne Umgebung. Wirtschaftsinformatik. dpunkt.verlag

SoftwareAG (2017) ARIS Method Manual v.10

Terlouw L. I., Albani A. (2013) An enterprise ontology-based approach to service specification. In: IEEE Transactions on Services Computing 6(1), pp. 89–101

The Open Group (2013) ArchiMate 2.1 Specification: Open Group Standard. The Open Group Series. Van Haren, Zaltbommel

Warmer J. B., Kleppe A. G. (2003) The Object Constraint Language : getting your models ready for MDA, 2nd ed. Addison-Wesley, Boston

Wettinger J., Breitenbücher U., Leymann F. (2014) Standards-based DevOps automation and integration using TOSCA. In: IEEE/ACM 7th International Conference on Utility and Cloud Computing. IEEE Computer Society, Washington, DC, USA, pp. 59–68

Winter A., Ziemann J. (2007) Model-based Migration to Service-oriented Architectures. In: Softwaretechnik-Trends 27(1)

Yu E. (1997) Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. In: 3rd IEEE International Symposium on Requirements Engineering. RE 1997. IEEE Computer Society, Washington, DC, USA, pp. 226–236

Zhang L. J., Zhou N., Chee Y. M., Jalaldeen A., Ponnalagu K., Sindhgatta R. R., Arsanjani A., Bernardini F. (2008) SOMA-ME: A platform for the model-driven design of SOA solutions. In: IBM Systems Journal 47(3), pp. 397–413

Ziemann J., Leyking K., Kahl T., Werth D. (2006) SOA Development Based on Enterprise Models & Existing IT Systems. In: Cunningham P., Cunningham M. (eds.) Exploiting the Knowledge Economy Issues, Applications, Case Studies. IOS Press, pp. 83–90

Zimmermann O. (2017) Microservices Tenets. In: Journal of Computer Science 32(3-4), pp. 301–310

Zimmermann O., Milinski S., Craes M., Oellermann F. (2004) Second generation web services-oriented architecture in production in the finance industry. In: Companion to the 19th Annual ACM SIGPLAN OOPSLA Conference. ACM, pp. 283–289