Visual Description of Digital IT Consulting Services Using DITCOS-DN: Proposal and Evaluation of a Graphical Editor

Meikel Bode(✉), Maya Daneva, and Marten J. van Sinderen

Department of Semantics, Cybersecurity and Services, University of Twente, Drienerlolaan 5, 7522 NB Enschede, The Netherlands
{m.bode,m.daneva,m.j.vansinderen}@utwente.nl
https://www.utwente.nl/en/eemcs/scs

Abstract. The digital transformation of the IT consulting domain recently gained momentum due to the Covid-19 pandemic. However, the range of IT consulting services that are fully digital is still very limited. Plus, there are no standardized and established methods for describing digital IT consulting services, nor there is any suitable tooling for digital IT consulting service provisioning. The present work aims to reduce this gap by contributing to establishing a well-defined approach to formally describing digital IT consulting services that could possibly be a candidate for standardization. Building upon (i) the ontology DITCOS-O, which provides the semantic basis for our approach, and (ii) the YAML-based description notation DITCOS-DN, which we leverage to describe digital IT consulting service models, we propose a graphical, web-based editor (called DITCOS-ModEd) to simplify service model maintenance. Following a design science based research process, we developed a prototype and empirically evaluated its applicability with the help of IT consultants. This first evaluation allowed us to identify some limitations and to plan specific improvements, both to the underlying artifacts DITCOS-O and DITCOS-DN, as well as to DITCOS-ModEd itself.

Keywords: Digital transformation · IT consulting · Service ontology · Service description · Virtualization · Consulting platform · Graphical editor · DITCOS-O · DITCOS-DN · DITCOS-ModEd · Design science · YAML

1 Introduction

During the Covid-19 pandemic the digital transformation (DT) of the IT consulting (ITC) domain gained momentum. The core of the ITC business is to advise clients on how to digitally transform their respective processes and businesses models. Gartner defines ITC services to be "...advisory services that help clients assess different technology strategies and, in doing so, align their technology strategies with their business or process strategies. These services support customers' IT initiatives by providing strategic, architectural, operational and implementation planning" [7, term: 'it consulting'].

In this work we use the terms digitization, digitalization, and digital transformations in line with the definitions of Gartner. Digitization is the conversion of physical resources to digital representations; digitalization is the use of digital resources within IT systems; and digital transformation refers to digitalization in the context of business processes in ITC [7, terms: 'digitization', 'digitalization', 'digital transformation'].

However, digital ITC services and suitable tools for the digital provision of these services are still rare. We use the term *digital ITC service* as defined in [3]: "Digital IT consulting services are technology-based consulting services represented by standardized, modularized, re-combinable, reusable, and customizable service assets that carry specific service commitments and are provided either in an automated, hybrid, or manual mode by human and/or technical agents or in a self-service manner and are instantiated, delivered, monitored, and orchestrated by digital consulting platforms." [3].

In the literature of digital ITC, to the best of our knowledge there are hereonly a small number of examples, such as the *eConsulting Store* provided by Werth et al. (2016) and the customer-tailored web-based self-service project assessment solution provided by Nissen et al. (2019) [10,13]. These artifacts have in common that they are tailored to a special problem context and do not aim on solving issues such as semantic standardization, modularization, or reusability as we identified to be necessities regarding the digital transformation of ITC [3].

To address these gaps in our recent research, we have lately contributed two artifacts [4]: (1) DITCOS-O, an ontology covering relevant concepts of digital ITC, and (2) DITCOS-DN, a formal description notation based on DITCOS-O providing a YAML syntax for service model definition. DITCOS-O constitutes an ontological sound basis to build upon. DITCOS-DN consumes these concepts and acts as a kind of 'programming language' to define ITC service models that are understandable for humans as well as interpretable by technical systems, such as digital ITC platforms.

With the current paper we build upon the published artifacts [4] DITCOS-O and DITCOS-DN and contribute a new artifact, namely the editor DITCOS-ModEd that supports end users to easily describe DITCOS-DN-based service models web-based and graphically. Our editor was empirically evaluated for suitability and usefulness by means of a two-step evaluation study, including perception-based research and experiments.

The remaining paper is structured as follows. Section 2 presents our research goals. Section 3 is on background and related work. Section 4 is on our research process. Section 5 describes our application of design science in order to create and evaluate our prototype editor. Section 6 discusses our results and Sect. 7 concludes.

2 Research Goals

As stated in the introduction, this article leverages our previously published results [4], namely the ontology DITCOS-O and the description notation DITCOS-DN. Both were empirically evaluated [4] with practitioners in a study that investigated the understandability of the YAML-based DITCOS-DN notation. Our evaluation results indicated that manual maintenance of DITCOS-O models using DITCOS-DN is well-supported by existing tools, since we assured automatic syntax checking, code completion and code formatting by means of integrated development environments (IDEs). However, during our evaluation it became apparent that even if good tool support for the textual creation of DITCOS-O service models described in DITCOS-DN exists, this might not be the preferred method from the practitioners' point of view. In fact, it became clear to us that practitioners would much more prefer working with a visual service description editor over using a tool for textual descriptions. Moreover, by means of a graphical editor, we believe we will create an important scientific instrument, which will be extremely useful and necessary for us in the context of our short-term future research activities especially in collaboration with practitioners. With this in mind, we set out the following goals with the present work:

1. To provide a graphical editor for the maintenance of DITCOS-O service models to be described in DITCOS-DN.
2. To apply experimentally the graphical editor in a real-world context with the participation of practitioners and to evaluate the suitability and usefulness of the solution in order to collect feedback for improvement of the underlying artifacts DITCOS-O and DITCOS-DN, as well as the graphical editor DITCOS-ModEd.

3 Background and Related Work

There are two streams of related work that are relevant for this paper: publications on service description approaches and on graphical service modelling. Regarding service description, the existing approaches are of two types: ontology-based and textual approaches using frameworks like IT Infrastructure Library (ITIL). As this research adopts an ontology-based approach, in what follows we provide related work concerning approaches of this type. One example approach is LinkedUSDL [5]. LinkedUSDL (Universal Service Description Language) was designed as an *upper ontology* with the aim to cover all relevant service contexts and concepts [5]. It uses the Resource Description Framework (RDF) to describe concepts based on *triples*, using the structure $subject \rightarrow predicate \rightarrow object$. LinkedUSDL follows the linked data principles [6] in the sense that it requires each element of the triple to be an Uniform Resource Identifier (URI), which, in the optimal case, points to additional content related to the respective RDF element. LinkedUSDL was organized into different sub-ontologies that complement and build upon each other.

Next, regarding graphical service modeling, an example is OBELIX [1], an ontology-based approach that helps to describe real-world services and service bundles based on the flow of resources and generated value (value webs). OBELIX defines *service elements* that have input and output *interfaces*, each of which supports an arbitrary number of *ports*. Output ports connect to input ports, while multiple service elements could form *service bundles*.

While searching for related work for the purpose of this research, we found that even though numerous approaches exist in the literature on technical service or real-world service description, we were not able to find approaches dedicated to the description of digitalized real-world ITC services provided through digital consulting platforms. We noticed that either the approaches aim on being as generic as possible (LinkedUSDL) and require complex RDF based descriptions or mainly focus on value flow, such as OBELIX.

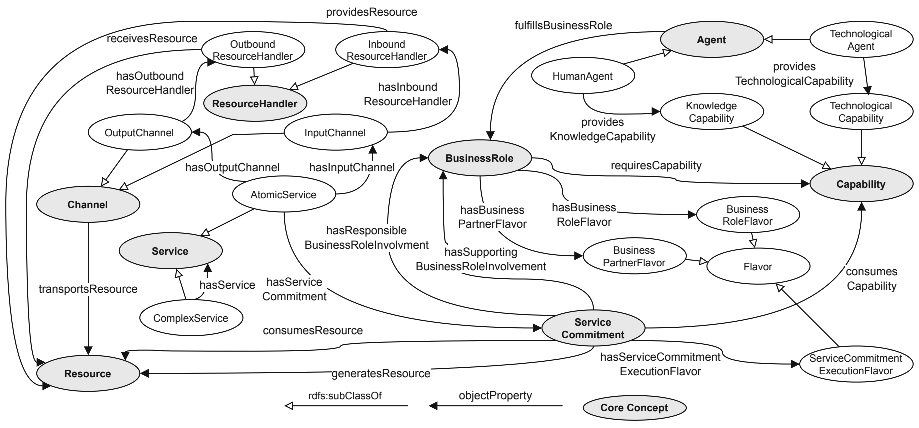


Fig. 1. The DITCOS-O and DITCOS-DN core concepts

Unlike the existing ontology-based approaches, with DITCOS-O and the corresponding YAML-based description notation DITCOS-DN, we want to provide a notation that aims on covering relevant core concepts of the ITC domain (see Fig. 1 how the core concepts relate), being easy to learn, to understand, and to use by practitioners (IT consultants), and is at the same time interpretable by systems (see Listing 1 for an simplified, reduced and invalid example. For a full example see [4]). It is meant to describe ITC services that are digitally provided through digital consulting platforms. The provisioning process might be either (1) only aided by helping agents, e.g., IT consultants, to conduct certain activities, (2) orchestrate incorporated agents, such as consultants, clients, or technical systems, or (3) fully automate the service provisioning process. To the best of our knowledge, none of the approaches in the literature covers this.

```
ditcosModel:
  metadata: [id, name, version, description, author, entryService]
  businessRoles: [businesRoleA, businessRoleB, ...]
  resources: [resourceA, resourceB, ...]
  services: [atomicServiceA, ..., complexServiceA, ...]
```

Listing 1. Simplified Example of a DITCOS-DN Service Model

4 Our Research Process

This section explains and motivates the process used to develop the DITCOS-ModEd artifact. We note that this work is part of a larger research project and builds upon already contributed artifacts [4]. Our research process adopted the design science (DS) research methodology of [11]. We chose it, because DS is recommended to research contexts such as ours where solutions (called 'artifacts') are designed to counter industry-relevant problems and issues [11]. Following Peffers et al. [11], our research process consists of the stages: (1) problem identification, (2) definition of objectives, (3) design and development, (4) demonstration, (5) evaluation, and (6) communication. In the following section we report on our execution of these stages, except stage six which is covered by the overall paper implicitly.

5 Designing the DITCOS-ModEd Graphical Editor

5.1 Problem Identification

In our recently published work [4], we already created and evaluated DITCOS-O service models using the YAML-based notation DITCOS-DN. These kind of service models are stored in a *service repository* (SR) to be later consumed by a digital ITC platform [2]. A central part of the platform should be an editor that supports textual (YAML-based) as well as graphical creation of DITCOS-O service models described using DITCOS-DN (see Fig. 2). We emphasize that the SR and the digital ITC platform are subject of our immediate future research.

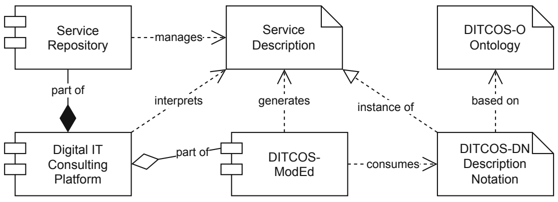


Fig. 2. Interplay of our research artifacts

Even though graphical service model creation is not technically required, as all descriptive power is provided by DITCOS-DN, the practical application of the textual modeling approach remains more complex and technical as it needs to be. Hence, we decided to provide the possibility to graphically create DITCOS-DN-based service models with the help of an appropriate, web-based editor. By providing this graphical editor, we also aim to increase its possible adoption by practitioners who may not be familiar with programming languages and have more of a business focus. It is worthwhile noting that our decision to provide a graphical and web-based editor component is also consistent with the architectural requirements identified in our previous exploratory study focused on practitioners' requirements elicitation [2]. We refer to a subset of these requirements in column 'Source' in Table 1.

5.2 Definition of Design Objectives

Our two goals for this research were stated in Sect. 2. Linked to them, we defined the following objectives in form of functional and non-functional requirements [9]. These requirements listed in Table 1 are then later to be used during the evaluation of our newly proposed editor. We assigned to each requirement in Table 1 an identifier (ID), a name, a description, a type (either functional or non-functional), and the source we collected the requirement from.

Table 1. Functional and non-functional requirements for DITCOS-ModEd

ID	Name	Description	Type	Source
R1	Web-based	Build the editor using web-based technologies	NF	[2]
R2	Component-based	Design the editor to be built from reusable components	NF	
R3	External Lookups	Enhance JSON-Schema based value lookups to support web-services	F	[4]
R4	CRUD Support	Support creation, reading, updating, and deleting service models	F	[2]
R5	Graphical View	Support the graphical rendering of service models and component interconnection	F	
R6	YAML View	Support the YAML code inspection	F	
R7	DITCOS-O Coverage	Support concepts of the DITCOS-O ontology expressible in DITCOS-DN description notation, such as Atomic Service, Complex Service, Business Role, Capability, Resource, Service Commitment	F	

5.3 Design and Development

This section briefly describes the architecture and the design we chose for our proposed editor as well as the tools we used for development. Below we present the elements included in our design and the choices we made in regard to each element.

Editor Architecture. Based on the software design pattern model-view--controller (MVC) [12] we structure the editor artifact into three layers: (1) presentation, (2) business logic, and (3) data access. The data access layer consumes an externally provided data services layer. These layers will be discussed in the next sections in more detail. Overall, the graphical editor will be implemented as a standalone prototype application that is not yet embedded into a larger architecture as we depicted in [2]. The reason for this is, that we mainly want to test the graphical modeling approach of DITCOS-DN and the generation of valid YAML-based service descriptions. The integration of DITCOS-ModEd into the overall architecture will therefore be covered in our future research.

Presentation Layer. Important requirements on the graphical editor are web-based and component-based design. Today, web-applications follow either the backend-rendering, the frontend-rendering, or a hybrid approach. For our implementation we decided to go for the frontend-rendering approach using the REACT framework, because it is a recent framework and the main author already has experience using it. We checked other options, such as Vue or AngularJS but decided against it as this would have meant to learn a new framework.

The loading and execution of a REACT application is initiated after the page that references the REACT JavaScript-based logic was transferred to the browser. The JavaScript logic then renders the entire web-application document object model (DOM) dynamically and manages subsequent page updates and data acquisition by executing HTTP requests on demand. REACT aims on the design and implementation of simple and independent components which can be combined to build more complex applications. The REACT components render out HTML5 compliant code by directly manipulating the DOM. For styling the HTML elements cascading style sheets (CSS) are being used. We used the common and widely-used web component theme 'Material Design' for our REACT components. We decomposed the prototype into 17 independent REACT components that work together. All components are described in Table 2.

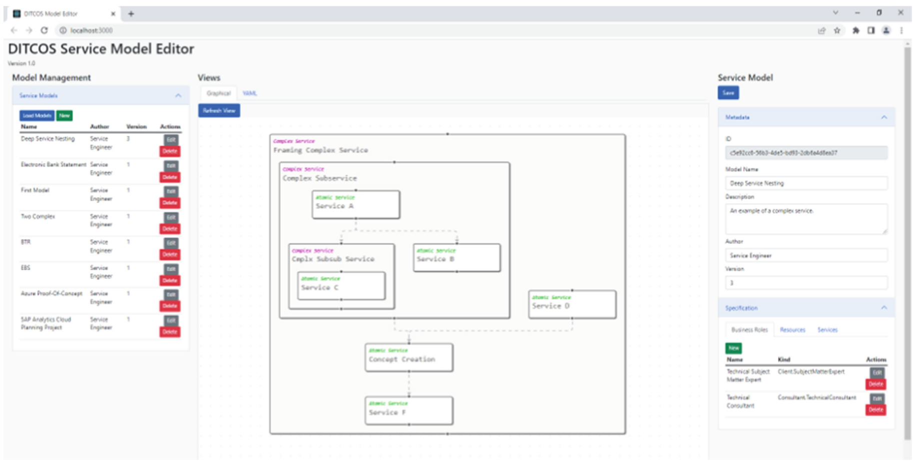
Business Logic Layer. Next to the presentation layer and the associated presentation logic, REACT applications contain their required business logic. This is due to the fact, that the web-application is self-contained. All application logic gets loaded during the initialization phase after the stub of the application has been transferred from the web server to the browser. Upon initialization, the REACT App component (see Table 2) gets executed, which itself triggers initialization and execution of all other components in the REACT component hierarchy. All component executions load their required data either individually or shared from remote sources, which are usually provided by the server that also serves the web-application stub and its logic. Additional business logic might be externalized to the server side and exposed by web-based API to consuming web-applications, such as duplicate checks or other kind of data validations before being persisted to a store. This kind of APIs often provides common or shared functionality that can be used by different (web) applications.

Table 2. REACT components of DITCOS-ModEd

Name	Description
App	Represents the central component of an REACT application. It embeds the *ModelManager*, *ModelView*, and the *ModelEditor* top-level components
ModelManager	A top-level component that represents the interface to existing DITCOS-O service models and supports their creation, deletion, and loading
ModelViews	A top-level component that embeds the components *GraphicalView* and *YAMLEditor*
ModelEditor	A top-level component that embeds the components *ModelMedatadata*, *BusinessRoles*, *Resources*, and *Services*. It provides the functionality to persist the currently loaded model
GraphicalView	Provides the graphical representation of the defined DITCOS-O service model currently loaded in form of a directed graph consisting of nodes and edges. It is based on the external REACT component *REACTFlow*. It supports interactions such as zooming in and out, shifting the graphicalized graph around, the selection of nodes and edges. It also provides a *MiniMap* component that represents a minimized version of the whole graph for quick navigation
YAMLEditor	Provides the textual representation of the currently loaded DITCOS-O service model. It is based on the external REACT code editor component *Monaco*. It supports line numbering, folding, formatting, coloring, and indent
ModelMedatadata	Represents input elements for model metadata, such as id, name, description, author, and version
BusinessRoles	Represents all business roles defined by the model and supports their creation, deletion, and modification by linking to *BusinessRoleDetailsDialog* component
BusinessRole-DetailsDialog	Provides input elements to model a business role and link to the capabilities that constitutes it
Resources	Represents all resources defined by the model and supports their creation, deletion, and modification by linking to *ResourceDetailsDialog* component
ResourceDetails-Dialog	Provides input elements to model a resource generated, consumed, or updated by service commitments of defined atomic services
Services	Represents all atomic and complex services defined by the model and supports their creation, deletion, and modification by linking to *AtomicServiceDetailsDialog* and *ComplexServiceDetailsDialog* components
AtomicService-DetailsDialog	Provides input elements to model an atomic service and its service commitments by linking to *ServiceCommitmentDetailsDialog* component
ServiceCommit-mentDetailsDialog	Provides input elements to model a service commitment and to refer to resources by linking to *InvolvedResourcesDetailsDialog* component
InvolvedResource-DetailsDialog	Provides input elements to refer to a certain defined resource and its type of involvement in the related service commitment
ComplexService-DetailsDialog	Provides input elements to model a complex service and to link to its constituting atomic and complex services by linking to *InvolvedServiceDetailsDialog* component
InvolvedService-DetailsDialog	Provides input elements to refer to a certain defined atomic or complex service and to define its dependencies on other services

Data Access Layer. The data access of REACT applications usually is realized by consuming REST-based APIs exposed by the server side. Other HTTP-based communication protocols, such das GraphQL or OData are also common. The data exchange format can vary, but nowadays mostly JSON formatted messages are interchanged with incorporating the data service layer. For our editor we chose a combination of REST-API and JSON as message format.

Data Service Layer. The data services used by our prototype are provided by a JavaScript based server component. All data entities used are represented by corresponding REST-API endpoints which support the CRUD pattern by incorporating different HTTP methods, such as GET, PUT, PATCH, and DELETE. For our prototype we use a simple server component without applying extended validations. This is sufficient, as building such an API would be delegated to a dedicated service repository which would be part of a larger digital ITC platform architecture as proposed in [2].



Name	Author	Version	Actions
Deep Service Nesting	Service Engineer	3	Edit Delete
Electronic Bank Statement	Service Engineer	1	Edit Delete
First Model	Service Engineer	1	Edit Delete
Two Complex	Service Engineer	1	Edit Delete
BTR	Service Engineer	1	Edit Delete
EBS	Service Engineer	1	Edit Delete
Azure Proof-Of-Concept	Service Engineer	1	Edit Delete
SAP Analytics Cloud Planning Project	Service Engineer	1	Edit Delete

ID: c5e92cc6-56b3-4de5-bd93-2db6a4d8ea37
Model Name: Deep Service Nesting
Description: An example of a complex service.
Author: Service Engineer
Version: 3

Name	Kind	Actions
Technical Subject Matter Expert	Client.SubjectMatterExpert	Edit Delete
Technical Consultant	Consultant.TechnicalConsultant	Edit Delete

Fig. 3. DITCOS-ModEd showing the available service models and the graphical view as well as the details of the currently loaded service model (left to right).

5.4 Proof-of-Concept Study Demonstration

We set up a proof-of-concept study [14] to explore the practical applicability of our proposed editor from the perspective of IT consultants. The underlying motivation for this was to have a very first prototype demonstration and collection of first feedback from practitioners. We wanted to assure that our prototype design work goes in the right direction and that the editor matches the possible expectations of practitioners. To this end, DITCOS-ModEd was demonstrated by the first author to five practitioners. Each joined a one-on-one 60 min long session with the researcher. The five participants in our proof-of-concept study are all working for a midsize IT consultancy in Germany. Three were technical experts, while two were business process analysis experts with relatively little technical background. The areas in which the five participants were consulting their clients varied from SAP Finance, SAP Business Intelligence, SAP Logistics, Business Strategy, and Microsoft Azure Cloud.

Proof-of-Concept Session Setup. We prepared the editor and run it locally at the computer of the first author using the IDE built-in features. Every session started with an empty and clean editor instance.

Session Execution. We conducted all sessions remotely via Microsoft Teams due to the lockdown work-from-home policies. We prepared a note form which we used to take notes during the each session. For all sessions we executed the following process described in Table 3.

Table 3. Session execution phases

Phase and Description
Phase 1 (~5 min) We welcomed the practitioner and explained what are the aims of the session are and how we will proceed
Phase 2 (~15 min) We introduced DITCOS-O and DITCOS-DN shortly to the practitioner to make him familiar with the concepts of ITC service description. In particular the concepts of a DITCOS-O service model, such as atomic vs. complex service, service commitments, business roles, capabilities, and resources (see Fig. 1)
Phase 3 (~10 min) We introduced the practitioner to the DITCOS-ModEd editor and referred to the concepts presented in phase 2. Furthermore, we explained the user interface and its functionality to the practitioners. Using an example DITCOS-O service model, we showed how the previously introduced concepts can be modeled using the editor and how the graphical modeling reflects to the resulting YAML-based DITCOS-DN textual service description
Phase 4 (~25 min) We asked the practitioner to model a selected ITC service using the editor. This was done with the help the first researcher concerning the use of the editor's interface and navigation. Each expert did the conceptual modeling work himself, while the researcher took over the handling of the interface. This was since the session would have required a much longer training of the practitioner at this early stage of the editor's development. During this phase we continuously discussed the current state of the DITCOS-DN model and inspected the resulting YAML code. During this very short phase of 25 min, the aim was to give the practitioner a feeling for creating DITCOS-O service models using the editor with the goal of gathering his feedback which we would consider for inclusion in our future design cycles of the editor. A complete realization of a DITCOS-O service model would not have been possible within the time frame of 25 min
Phase 5 (~5 min) This phase was to wrap-up and let the practitioner tell us about his experience with the editor. We asked for feedback related to (i) the graphical user interface (GUI), (ii) the usability and concepts, and (iii) general suggestions and comments on possibilities to extend the prototype. The last included possible 'nice to have' integration to other tools that consultants use. In addition to that, we also noted our own (researcher) suggestions and observations during the sessions

5.5 Collected Feedback and Suggestions

All five of the practitioners liked the representation and the Look&Feel of the editor. One had the impression that it supported all the relevant DITCOS-O concepts well (see Fig. 1). Another practitioner stated that the interaction and working process with the GUI was particularly fluent. In addition, we received seven feedback items (see Table 4) to our notation concepts and components of the editor and four suggestions (see Table 5) for improvement of the artifacts.

Table 4. Feedback on usability and concepts

ID	Concept/Component	Name and Description
F1	AtomicServiceKind, BusinessRoleKind, Capability	**Insufficient Coverage** Realistic modeling requires more choices of the named concepts. Add more kinds of each concept for subsequent experiments
F2	BusinessRole	**Agent Occupation Mode** Add an attribute to the business role concept that indicates the 'agent occupation mode' to indicate that a certain role must be fulfilled by a dedicated agent that cannot have other roles assigned within the same service commitment
F3	BusinessRole	**Capability Level** Modeling levels of a capability is not possible in the sense that a certain business role requires 'SAP FI Senior' and 'SAP ABAP Intermediate' skill levels. Actually one can only define the overall level of a business role by assigning the capability 'Knowledge. Experience.(Junior, Intermediate, Senior)'
F4	BusinessRole	**Decision Power** Modeling of 'decision power' of a business role is not possible. This might be relevant if a business role must make decisions during the service provisioning. This flag seems to be relevant in cases necessitating the modeling of client-side roles. *Example*: A client subject matter expert is required to decide to go for a certain customization variant related to the chart of accounts
F5	ServiceCommitment	**Inline Definition of Concepts** The current version of the editor requires that the service modeler defines all resources and business roles before starting with the modeling of service commitments of atomic services. This does not feel 'natural' because, sometimes required resources or business roles only become obvious during the service modeling itself. In the current editor's version, the user would have to abort and step out of the service modeling and create additional resources or business roles and then re-enter the service modeling
F6	ServiceCommitment	**Planned Duration** Add 'planned duration' as attribute to the service commitment concept. Based on this, it would become possible to calculate estimated time consumption for a DITCOS-O service model. (Added by researcher)
F7	GraphicalView	**Drill Down** Add a recursive drill down functionality to the service nodes

5.6 Reflection on How the Editor Reaches Its Goals and Requirements

An important part of the DS research process, according to Peffers et al. [11] includes an analytical reflection on how the designed artifact meets its goals set at the beginning of the research. This section reports our analytically reflection on the results obtained throughout the design of our editor and the proof-of-concept evaluation. The reflection is in two regards: (1) how the editor meets our main research goals defined in Sect. 2, and (2) how it meets the additional design objectives formulated in Sect. 5.2 as requirements.

Table 5. Suggestions and comments regarding nice-to-have extensions

ID	Name and Description
S1	**BPMN Diagram** Add a BPMN diagram to graphicalize a DITCOS-O service model as BPMN process
S2	**Gantt Diagram** Add a Gantt diagram to graphicalize the sequence/parallel dependencies of a DITCOS-O service model
S3	**Integrated Help System** The editor requires an integrated help system to guide the service modeler continuously
S4	**Service Reuse** Avoidance of duplicated names for model entities. This is important when services shall be reused in complex services
S5	**Tags** Allow arbitrary tags for all model concepts to assign non-service-related information. Predefine tags to avoid duplicates

Results Pertaining Our Research Goals. We formulated two main research goals. We present the results in the next paragraphs:

Provide an Editor for DITCOS-O Service Models. We aimed on delivering an editor to graphically create DITCOS-O models defined in DITCOS-DN. We were able to achieve this goal. DITCOS-O services models created using the DITCOS-ModEd editor cover all DITCOS-DN concepts, such as atomic and complex service, service commitment, business role, capability, and resource. We checked the completeness and syntactic correctness for each of the mentioned concepts as well as the interaction of the concepts in the context of a DITCOS-O service model.

Proof-of-Concept Evaluation of the Artifact. We presented the DITCOS-ModEd editor to five practitioners in an experimental setting where each session last around 60 min. Beside an introduction to the DITCOS-O, DITCOS-DN artifacts, as well as the editor, we modeled jointly with the practitioner an ITC service, without the aim of being complete. Our goal was to utilize the features and functionalities of the editor and not to create a fully elaborated DITCOS-O service model. During the modeling process we constantly discussed the activity and collected feedback as well as suggestions from the practitioners.

Results Pertaining the Design Objectives

R1: Web-based. We built DITCOS-ModEd using REACT, one of the latest JavaScript frameworks to build recent web-applications. REACT applications are self-contained in sense of logic and consume web-based APIs, such as REST. With our artifact we were able to fulfill this requirement.

R2: Component-based. Our artifact is based 17 independent and reusable REACT components. All components consume a common data layer, that represents the currently active DITCOS-O service model expressed in DITCOS-DN. All components interact with their respective part of the DITCOS-DN description. This requirement could be fulfilled.

R3: External Lookups. This requirement is twofold. First, we provide certain value completion functionality to the different REACT components. Examples for value helps are AtomicServiceKinds, BusinessRoleKinds, ResourceKinds, or Capabilities (see Fig. 1). If a certain component gets activated it consumes an external REST API, that provides e.g., a constant list of capabilities as a response. This list gets consumed by the REACT component to provide a value help to the user. This first perspective could be fulfilled. Second, another perspective relates to dynamically created instances of concepts, such as atomic or complex services, business roles, or resources. Typically, these instances would have been created by other users and within other DITCOS-ModEd instances. They would have been persisted in a central service repository and provided to DITCOS-ModEd by appropriate REST-APIs. This functionality will only be supported by a later prototype of DITCOS-ModEd. The reason is, that we are at an early stage of the artifact development. Hence, our focus is on the required basic functionality to support the creation of DITCOS-O service models. A later prototype will support reuse of defined concept instances, what was also suggested (S4) by a practitioner (see Sect. 5.5). Overall we were able to fulfill this requirement partly.
R4: CRUD Support. With the current prototype of DITCOS-ModEd we fulfill this requirement for the following DITCOS-O primary concepts: atomic and complex service, service commitment, and business role. For these concepts we created dedicated REACT components, that support creation, update and delete completely (see Table 2). The existing components enable users to create feature-complete DITCOS-O service models. Nevertheless, with the current prototype we decided not to support CRUD for instances of supporting concepts, such as AtomicServiceKind, BusinessRoleKind, Capability, ResourceKind, ResourceInvolvementType MIMEType, ServiceCommitmentKind, and ServiceCommitmentExecutionFlavor (see Fig. 1). Instead, we decided to focus only on the creation of DITCOS-O service models and the required primary concepts and to assume the existence of the named supporting concepts. We simply provide instances of the supporting concepts in form of constant lists, as already described. In other words, for these supporting concepts, we only provide 'read' functionality with the current DITCOS-ModEd prototype. In light of this discussion, we could say that this requirement could be only partly fulfilled.
R5: Graphical View. With the current prototype we where able to provide a sophisticated graphical view on DITCOS-DN based DITCOS-O services models already. The view supports nodes, edges, zooming, change of viewport just to mention some features. Related to DITCOS-O service models anyhow, we only support a subset of the concepts with the current prototype. These are namely atomic and complex services as well as the edges to connect subsequent concepts (see Fig. 1). Nevertheless, the current prototype of the editor can provide a graphical representation of the two core concepts and also supports the required visualization of hierarchical organized complex services, which may contain sub-services which themselves could be complex services again. Based on this reasoning, we think that overall we fulfilled this requirement partly.

R6: YAML View. The current prototype of DITCOS-ModEd supports the complete rendering of DITCOS-DN based DITCOS-O service models in YAML representation. The YAML viewer supports code formatting, code-indent, and coloring. The current implementation only supports one-way YAML rendering based on an in-memory representation of the corresponding DITCOS-DN. Changes to the YAML code therefore, do not trigger changes to the REACT components dynamically. As this was not our aim with this early prototype of DITCOS-ModEd, we fulfilled this requirement completely.
R7: DITCOS-O Coverage. With the current prototype of the DITCOS-ModEd we realized a DITCOS-O coverage regarding to DITCOS-DN based service models a 100% completeness. Therefore, we conclude that we achieved this requirement completely.

6 Discussion of the Findings in Our First Evaluation

Our proof-of-concept study with the ITC practitioners showed that the current prototype of DITCOS-ModEd, even though it is at early stage, already was recognized as a helpful tool that supports the modeling process of DITCOS-O service models compared to the pure textual modeling by manually coding the necessary YAML using DITCOS-DN description notation. The feedback on GUI and the Look&Feel of DITCOS-ModEd is clear (see Sect. 5.5). Anyhow, the feedback on concepts we collected, as well as our own experience with DITCOS-ModEd made clear, that the editor is not yet ready to be transferred to the consulting practice. While this was expected, our proof-of-concept study indicated that the proposed editor did meet our research goals and requirements (to a very large extent).

Our immediate future research that builds upon DITCOS-ModEd must explicitly respond to the feedback (F1) related to insufficient coverage regarding to the concepts of AtomicServiceKind, BusinessRoleKind, and Capability (see Sect. 5.5). These concepts are consumed during the modeling process at different stages. For the current prototype it was sufficient, to only provide a small set of instances of each concept, as we only aimed on supporting the general modeling approach and not to be complete. During follow-up empirical evaluation we will ensure that the set of instances of these concepts are more complete.

Feedback F1 has no direct impact on the underlying artifacts DITCOS-O and DITCOS-DN, as only instances of these concepts are required, and they can be created at runtime as required. However, feedback points F2 to F6 directly affect the concepts BusinessRole and ServiceCommitment. These points require changes to the underlying ontology DITCOS-O and adjustments related to the description notation DITCOS-DN.

Related to the DITCOS-ModEd, we collected one suggestion (F7) namely, to add a drill down functionality to the graphical representation of the underlying DITCOS-DN description. From our point of view this makes absolutely sense. During the design cycles we experimented with different levels of details we added to the view. To ensure clarity, we opted for a rather reduced view of the model concepts. A drill-down to display further relevant details seems very useful here.

The general suggestions we received where also very helpful and interesting. In particular, the improvement ideas to add a BPMN and a Gantt representation to the underlying DITCOS-DN representation of the DITCOS-O service model seem worthy of attention. The question arises, which further requirements and additions to DITCOS-O/DN would be necessary for this?

Even though our aim is to provide an easy-to-use service description notation, the concepts might not always be self-explaining. Adding an integrated help system (S3) to DITCOS-ModEd is a desirable goal.

The practitioners' suggestions to reuse defined services (S4) and to allow arbitrary tags attached to all concepts, are also very good. As one participant indicated, service reuse is considered very important. Anyhow, we suggest 'concept reuse' as a more general improvement. In future prototypes of DITCOS-ModEd we plan to support the reuse of concepts in the sense, that concepts defined by a certain user can be referenced from other concepts created by another user. A prerequisite to support this feature is to implement a service repository as depicted and described in our digital ITC platform architecture proposal [2]. We will evaluate how to tackle the feedback items F2-F7 and suggestions S1-S5 during future research.

Finally, we reflected on the limitations of our empirical findings. We involved five practitioners and clearly it might be the case that if we could have included many more, it would have been possible to collect feedback points different from those that we have now. However, we note that we selected participants with exposure in a variety of consulting areas (ranging from finance, to logistics to cloud). Their perceptions and experiences of the 60 min sessions overlapped, specifically regarding the mater that they liked the editor and thought it indeed filled a gap in the consulting practice. We think that it might well be possible that similar perceptions might come if we possibly include in our sessions other consultants who might share the context in which our participants work. Consultants working in client organizations in the same business sector, implementing the same technology and following the same consulting approach, might well have similar conceptual modeling experiences and similar needs of an editor such as our participants do. As Ghaisas et al. indicate, similar contexts might possibly create similar work experiences [8]. However, as we plan to create more prototypes of the editor, a more empirical and longer-term evaluation is needed to fully assess its applicability, usefulness, and usability in a real-world context. This is our priority in the future.

7 Conclusions and Future Work

With this work we contributed the web-based, graphical editor DITCOS-ModEd to create DITCOS-O service models described using the DITCOS-DN description notation. Following the DS methodology [11], we developed a prototype of the editor and evaluated it in a proof-of-concept experimental study with ITC practitioners. The latter revealed some limitations of DITCOS-ModEd and enabled us to collect valuable and interesting feedback and improvement suggestions to enhance both the underlying artifacts DITCOS-O and DITCOS-DN, as well as the editor DITCOS-ModEd itself. Our reflection on the limitations helped us formulate immediate future research steps towards creating the next and improved prototype of our proposed editor.

References

1. Akkermans, H., Baida, Z., Gordijn, J., Peiia, N., Altuna, A., Laresgoiti, I.: Value webs: using ontologies to bundle real-world services. IEEE Intell. Syst. **19**(4), 57–66 (2004). https://doi.org/10.1109/MIS.2004.35
2. Bode, M., Daneva, M., van Sinderen, M.J.: Digital IT consulting service provisioning – a practice-driven platform architecture proposal. In: 25th IEEE International Enterprise Distributed Object Computing Workshop. The Gold Coast, Australia (2021). https://doi.org/10.1109/EDOCW52865.2021.00056
3. Bode, M., Daneva, M., van Sinderen, M.J.: Characterising the digital transformation of IT consulting services - results from a systematic mapping study. IET Softw. (2022). https://doi.org/10.1049/sfw2.12068
4. Bode, M., Daneva, M., van Sinderen, M.J.: Describing digital IT consulting services: the ditcos ontology proposal and its evaluation. In: 2022 IEEE 24th Conference on Business Informatics (CBI), Amsterdam, The Netherlands (2022). https://doi.org/10.1109/CBI54897.2022.00029
5. Cardoso, J., Pedrinaci, C.: Evolution and overview of linked USDL. In: Nóvoa, H., Drăgoicea, M. (eds.) IESS 2015. LNBIP, vol. 201, pp. 50–64. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-14980-6_5
6. Frank, A.G., Mendes, G.H., Ayala, N.F., Ghezzi, A.: Servitization and Industry 4.0 convergence in the digital transformation of product firms: a business model innovation perspective. Technol. Forecast. Soc. Change **141**, 341–351 (2019). https://doi.org/10.1016/j.techfore.2019.01.014
7. Gartner: Gloassary. https://www.gartner.com/en/information-technology/glossary (2022)
8. Ghaisas, S., Rose, P., Daneva, M., Sikkel, K., Wieringa, R.J.: Generalizing by similarity: lessons learnt from industrial case studies. In: 2013 1st International Workshop on Conducting Empirical Studies in Industry (CESI), pp. 37–42. IEEE, San Francisco, CA, USA (2013). https://doi.org/10.1109/CESI.2013.6618468
9. Lauesen, S.: Software Requirements: Styles and Techniques. Addison-Wesley, Boston (2002)
10. Nissen, V. (ed.): Advances in Consulting Research. CMS, Springer, Cham (2019). https://doi.org/10.1007/978-3-319-95999-3
11. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. J. Manag. Inf. Syst. **24**(3), 45–77 (2007). https://doi.org/10.2753/MIS0742-1222240302
12. Starke, G.: Effektive Softwarearchitekturen, 7th edn. Hanser, München (2015)
13. Werth, D., Greff, T., Scheer, A.W.: Consulting 4.0 - Die Digitalisierung der Unternehmensberatung. HMD Praxis der Wirtschaftsinformatik **53**(1), 55–70 (2016). https://doi.org/10.1365/s40702-015-0198-1
14. Wieringa, R., Daneva, M.: Six strategies for generalizing software engineering theories. Sci. Comput. Program. **101**, 136–152 (2015). https://doi.org/10.1016/j.scico.2014.11.013