An Experience Report on the Implementation of the KYKLOS Modeling Method

Georgios Koutsopoulos, Martin Henkel

An Experience Report on the Implementation of the KYKLOS Modeling Method

Georgios Koutsopoulos [0000-0003-2511-9086] and Martin Henkel [0000-0003-3290-2597]

Department of Computer and Systems Sciences, Stockholm University, Stockholm, Sweden
{georgios,martinh}@dsv.su.se

Abstract. Several types of enterprise models and methods have been developed that may help an organization to describe and improve its business. A common practice is also the development of tool support to complement an enterprise modeling method's application. The development of tool support for a modeling method includes creating a representation of the modeling concepts, but also designing how the user should interact with the tool. This paper reports on the challenges and opportunities encountered during the process of implementing the KYKLOS modeling method in a modeling tool. The KYKLOS method, which is an enterprise modeling method, is specialized in supporting the design and analysis of changing capabilities. Using as input an initial meta-model of capability change, all the necessary tasks are performed to elicit a language model, which is required for the implementation of the method in a tool.

Keywords: Enterprise Modeling, Meta-modeling, Implementation, Capability, Business Transformation

1 Introduction

Enterprise Modeling (EM), which is a subset of conceptual modeling, is focused on capturing organizational knowledge and providing input and motivation for the design of Information Systems (IS) for an organization [1]. The complexity of developing IS and other business solutions is on the rise because of rapidly changing business requirements [2]. The development and operation of an IS can be considered as a knowledge-based activity which is continuous and utilizes conceptual modeling in order to bridge the understanding of complex organizational phenomena and the effort to design IS which can support dynamic change and agility [3]. This usually involves employing modeling methods which have been implemented in supporting modeling tools. Using modeling tools to handle a method successfully is considered state-of-the-art, because they do not only support defining modeling languages and facilitate the creation of model representations that can be processed, but also enable accessing, storing and exchanging models and specifying functionalities for improved user experience [4]. A specialization of EM is capability modeling, which uses capability as its focal point. Several capability modeling methods exist and the majority also includes capability

modeling languages and notations. They usually employ different meta-models which consist of different sets of concepts to capture the nature of capabilities.

KYKLOS is one such method [5], designed specifically for designing and analyzing changing organizational capabilities. In order to capture the relevant characteristics, the phenomenon of capability change has been explored and conceptualized in the earlier steps of our study. Starting with identifying the relevant concepts already existing in the literature [6, 7], requirements were elicited [8] and the phenomenon of capability change was conceptualized in the form of an initial meta-model [9, 10].

To be readily useable for a modeler, the KYKLOS method was in need of tool support to aid the user in creating models of capability change. The implementation of the method required a modeling language meta-model, which used as input the initial meta-model. Therefore, during the implementation, several transformations were made to the initial meta-model. These transformations were done to make use of the tool platform, and to make the implemented language less complex. For example, several classes in the initial meta-model were implemented as attributes of other classes in the final tool implementation. Thus, the initial detailed conceptualization of the phenomenon of capability change and the conceptualization of the method bear significant differences, mainly because of different degrees of operationalization potentials existing among the meta-model's concepts.

The aim of this paper is to share the KYKLOS implementation experience with the Enterprise Modeling community by reporting the opportunities, challenges and lessons learned that have been encountered during the implementation of the method in a tool.

The rest of the paper is structured as follows. Section 2 briefly presents the related background. Section 3 provides an overview of the KYKLOS method's state before the implementation. Sections 4 reports on the implementation procedure and the included activities. Section 5 discusses the procedure and its results. Finally, Section 6 provides concluding remarks.

2 Background

The primary aim of conceptual modeling is the description of specific aspects of the physical and social world for understanding and communicating. An abstract representation of specific aspects of entities that exist within a specific domain is called a conceptualization, e.g. a meta-model, while an abstraction of the domain's state of affairs that is expressed via a conceptualization is called a model [11]. Since models are abstract entities, they must be represented using an artifact, for documentation, communication and analysis purposes, and this indicates the need for a highly expressive modeling language, the focus of which should be on representation adequacy [11].

Furthermore, to construct a model, guidance is needed in the form of a modeling method. As defined in [2], the components of a modeling method are a modeling technique, which consists of a modeling language and a modeling procedure, and mechanisms and algorithms working on the models that the language describes. A modeling procedure describes the required steps to apply the method to create the resulting model.

The modeling language consists of its syntax, semantics and notation. The *syntax* includes the description of rules and elements for the creation of models and is described by a grammar. The two major approaches for modeling language grammars are graph grammars and meta-models [2]. A common means to describe the meta-model of the syntax is by using UML class diagrams [12]. The *semantics* describe a modeling language's meaning, often using informal textual descriptions for semantic definition. The visualization of a modeling language is described by the *notation*. There are static approaches that define symbols for the visualization of the syntactical constructs, like pixel-based or vector graphics, yet these do not take into consideration the model's state. Dynamic approaches consider this state and usually split the notation in two parts; representation, which maps to the static approach, and control, which defines rules that influence the visualization according to the model's state [2].

An important factor for a successful modeling method and language is the provision of a set of modeling elements that can express the given domain abstraction, and this benefits from complementing efficient tool support [11]. Thus, there are specialized modeling tools that support the user in creating models that follow a certain syntax.

The domain specificity [13] of KYKLOS is organizational capability change. Since there is no consensus in the literature, the concept of capability is defined in this project as a set of resources and behaviors, with a configuration that bears the ability and capacity to enable the potential to create value by fulfilling a goal within a context [14]. Often considered as the missing link in business/IT transformation, it is associated to core business concepts like goal, resource, actor, process [15] and serves as the basis for change management, impact analysis and strategic planning [16]. A detailed review of the concept, and the variety of capability modeling approaches that exists in the literature has been explored and reported in an earlier step of this project [6].

3 Overview of KYKLOS Before the Implementation

KYKLOS, which has been introduced in [5], is a capability modeling method that focuses on capturing the concepts that are relevant to the phenomenon of capability change, aiming to support organizational change. This section describes the initial meta-model, and the modeling procedure, which consists of the required steps for applying the method.

Figure 1 shows the initial meta-model. Using the meta-model enables capturing changes in a given capability. The meta-model is based on a previously published framework [6], which includes the functions of change, in particular, observation, decision and delivery. Observation is captured using the concepts of context, which consists of monitored factors that are expressed as Key Performance Indicators (KPIs), and Intention elements. The model describes that a *capability* has at least one *configuration* that leads to the realization of the capability. *Resources* are allocated to a configuration which also consists of *behavior elements*. Realizing the capability produces at least one outcome, which can be measured to serve as criterion for a decision to change, along with the capability's assessment via contextual factors. Regarding delivery, it concerns the transition from a configuration to another. The meta-model also includes elements

of ownership in order to capture the owners of the capability, the change and the resources. When more than one owners exist, their interaction and boundaries are captured. One last part that is included in the meta-model is the description of the states of the capability and change in the form of their traits which have been identified in [7].

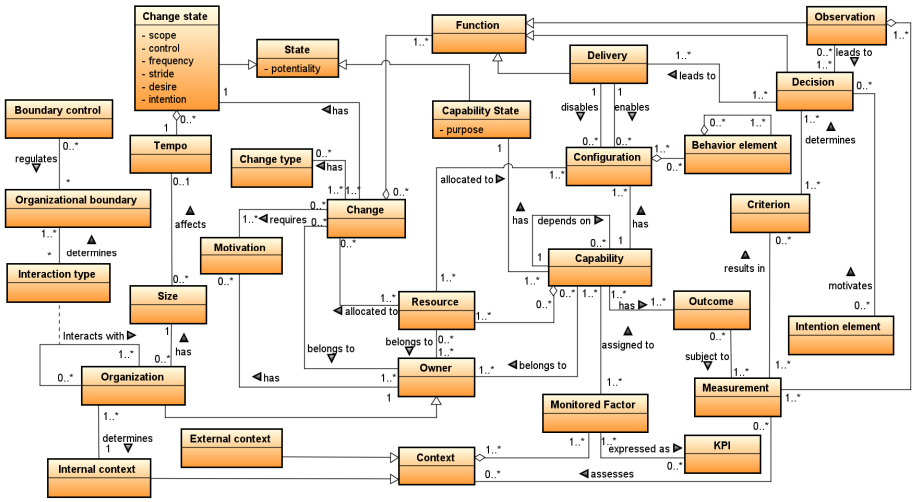


Fig. 1. The conceptualization of capability change, adapted from [10].

Concerning the modeling procedure, KYKLOS consists of four phases [5], namely (i) foundation, (ii) observation of context and intentions, (iii) decision alternatives, and (iv) delivery of change. Foundation is the initial phase and describes the base of the analysis, in terms of identifying the changing capability and its outcomes. The observation phase follows up and concerns capturing the need for change, in terms of context elements, with their associated monitored factors which are deemed relevant to the capability and are expressed as measurable KPIs. The organization's intentions are also captured in the form of goals, problems or any other element that motivates a change to the capability. The phase that follows is related to the analysis of alternative capability configurations that tackle the identified need to change. Different configurations can fulfill the same set of goals. A part of this phase is identifying the resources that need to be allocated to each alternative configuration and the behavior elements that are necessary for realizing the capability per configuration. Resources can be both material and immaterial. Capturing the ownership of the capability, the resources and change enables capturing potential organizational collaboration elements and the respective boundary controls. The final phase of the modeling procedure is the delivery of change. The focus of this phase is on understanding what is necessary to deliver the change. The change takes the form of a transition from one configuration to another. The delivery of change enables an inactive configuration while disabling an active one, or in the case of introducing a new capability or retiring an old one, there is a single enablement or disablement respectively. Describing how the change needs to be delivered includes capturing how the change is performed, in terms of identifying the attributes of change. These have been published in [7] and are (i) control, (ii) scope, (iii) frequency, (iv) stride, (v)

time, (vi) tempo, (vii) desire, and (viii) intention. The process can be iterative, if, for example, the delivery has an impact on the context or outcome of the capability, the initial phases can initiate again.

4 The KYKLOS Implementation

The implementation of the KYKLOS method in a tool requires using the initial meta-model to develop a language meta-model, developing a graphical notation and facilitating the user's interaction with the tool. The implementation has taken place using the ADOxx meta-modeling platform [17], which is provided by the Open Models Laboratory (OMiLAB). The use of a platform also meant that the implementation needed to use the ADOxx platform's concepts for model implementation. The implementation was done iteratively and involved the following steps:

- Conversion of the initial meta-model concepts to a language meta-model that could be implemented. This step included the decision if a concept should be represented as a concept, attribute, or relationship in the language meta-model. Moreover, several concepts were removed.
- Creating a syntax for the concepts in the tool meta-model. This included creating the graphical representation using the ADOxx GraphRep language.
- Creating tool behavior to facilitate user interaction. ADOxx is quite flexible, so it was possible to add several dynamic aspects to the model.

4.1 Initial meta-model to language meta-model conversion

A color-coded version of the initial meta-model is depicted in Fig. 2. The colors depict how they have been handled during the transition to the language meta-model. A detailed description of the process follows in the current section.

Conversion of Classes.
Transitioning from the initial meta-model to the language model provided the opportunity to reduce the number of included concepts. This contributes to reducing the complexity and clutter that has been identified to exist in the models derived from applying the initial meta-model [9]. The transition was done by converting initial classes to *attributes*, *association classes* or tool *functionalities*.

Conversion to Attribute.

- Owner: The Owner concept has been included in the initial meta-model to capture the ownership of capabilities, components and change. It has been modeled as a class as good modeling practice to avoid duplicate data. In the language model, it has been converted to an attribute with added functionality in the tool, which is better explained in Section 4.3. The introduced tool functionality makes it easier for the user to keep track of ownership, without the need to have it as a separate class.

– Tempo: This concept is a trait of change that has initially been modeled as a class because of its identified association to the Size class. Size is removed from the language meta-model (see below), and Tempo is converted to an attribute of Change.

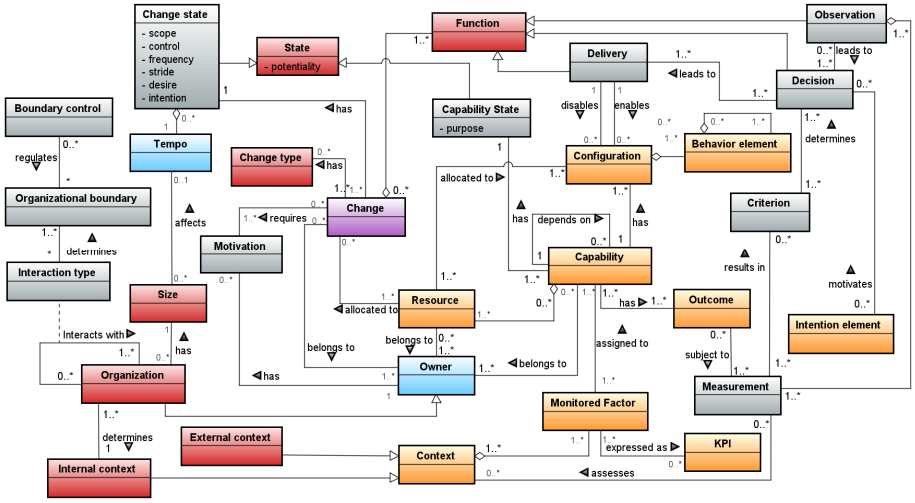


Fig. 2. A color-coded version of the meta-model, showing, remaining (orange) and removed (red) classes, along with classes converted to attributes (light blue), functionalities (grey) and association classes (purple).

Conversion to Association Class.

– Change: The concept of Change has been essential in the conceptualization and has been associated to various concepts, the majority of which do not exist as classes in the language meta-model. It was initially decomposed in three functions, observation, decision and delivery, but, since these are removed from the language meta-model and are implemented as method phases and tool functions, Change gains a link to the transitioning Configurations. Additionally, there are specific attributes of Change that need to be captured, therefore, it has been converted to an association class that describes how a configuration transitions to another, also gaining the Change State attributes.

Conversion to Tool Functionalities.

– Capability state: The concept of capability state is meant to capture whether a capability is active or not. This is captured in the tool by associating the capability to an active configuration. Therefore, the class can be omitted from the language model since the functionality of the tool will keep track of the active configuration.
– Change state: The class Change state existed to capture if a change is active or not. The tool version of KYKLOS can depict this via the existence of an active configuration element that is the target of change. The temporal attributes of Change can also assist. So, it can be omitted as well. Its attributes have moved to class Change.

- Observation: It is one of the three functions of capability change that has been modeled as a class in the initial conceptualization. Naturally, including a class that captures an activity bears value in a conceptualization but has limited utility in an implemented method and tool. As mentioned earlier, the KYKLOS method uses observation as its phase where the context factors and intentions whose fulfillment status motivates a potential change in the capability. All the necessary elements to perform this phase exist in the language meta-model, therefore, Observation can be omitted as a class.
- Decision: In a similar way to observation, the decision phase has been associated to a set of concepts that have been removed from the language meta-model, like criterion and configuration (as decision alternative), and replaced with tool mechanisms. The details on the specific related concepts follow.
- Delivery: The Delivery class captures the transition between capability configurations by enabling one and disabling another. In the method and tool implementation only one configuration is active at any time. This functionality captures the transition without a need to have the specific Delivery class.
- Criterion: This class refers to capturing how a decision is made, in terms of changing or not, and what to decide when changing. The tool design allows both these aspects to be addressed without including a specific Criterion class. Changing or not is motivated by the dynamic association elements between capability and contextual and intentional elements. In practice, a KPI or intention that is not fulfilled, is a criterion to change. What to decide refers to selecting a configuration among potentials. The tool allows a configuration to be active only when its required components are properly allocated. In this way, the decision is supported dynamically without needing the Criterion class, so, it is omitted.
- Measurement: In the initial meta-model, this class captured the act of comparing the target context and intention elements to reality. The functionalities described in the previous paragraphs also explain why this class has been omitted.
- Motivation: Same as Criterion, even though it can be included as an attribute to improve the descriptive ability of the tool. Moreover, in the implemented tool the motivation for performing a change can be implicitly shown by referring to one or several intention elements.
- Interaction type: This class captured the way two owning organizations interact with each other. The class requires a detailed understanding of the capability business ecosystem [18], which is not the primary goal of this project. The class has been converted to a high level tool functionality. The owners of the capability and the configuration components are captured in a control element of the notation that colors the borders of the components according to same or different ownership.
- Organizational boundary: Using the functionality that was introduced for different owners' interaction, the tool calculates the amount of externally owned required components and their owners and provides a decision-supporting suggestion to the user to take into consideration the reported results. In this way, the class is omitted from the language meta-model.
- Boundary control: Same as Organizational boundary.

Removal of Existing Classes.

- Size: It has been completely removed from the language meta-model. It refers to the size of an organization and has been introduced in [10], as a factor affecting the tempo of change. Even if an association between Size and Tempo has been strongly indicated, there was no clear and operational formula identified to provide utility in the tool. Thus, capturing the size of an organization without a clear effect on the tempo of change would have questionable value, therefore, Size was removed.
- Organization: As a specialization of Owner, the Organization class does not need to exist as a class since the parent class has been converted to an attribute.
- State: State existed in the phenomenon's conceptualization as a superclass of Capability state and Change state. There is no value in the existence of the superclass without its specializations, thus it is removed.
- Function: This class is the generalization of the three functions. Converting the specializations allows the removal of the superclass as well.
- Change type: This class captures if the change is an introduction of a new capability or the modification or retirement of an existing one. The model that is produced by the tool can capture this information by checking the activity states of configurations. If an active configuration has no prior alternative, it is an introduction, if it has transitioned from an alternative it is a modification and if it is deactivated without transitioning to an active configuration, it is retired.
- External context: The external context is a specialization of the Context class. The implementation can have a Context element described in terms of externality without a need for the specific subclass.
- Internal context: Same as External context.

Remaining Classes.
The remaining concepts of the conceptualization are the core elements and focal points not only of the KYKLOS method but also of the tool. They cannot be absent the language meta-model and they also retain their class status. The concepts included in define the fragment of the conceptualization that comprises the language meta-model are:

- Capability
- Configuration
- Outcome
- Resource
- Intention element
- Context
- Monitored factor
- Behavior element
- KPI

Introduction of New Classes.
The implementation provided the opportunity to introduce new classes to the language meta-model, as a means to improve the utility of the method via the tool. Three types of additions were performed to the KYKLOS meta-model in this step, in particular:

- Specializations of elements
  - The Behavior element, which is a meta-element, got a specialization class, in particular:
    - Process

The Process concept has been previously identified as the most common and popular concept [6] in the literature, regarding the behavioral aspect of capabilities. Other concepts like Activity, which are also popular, did not get included because a process consists of tasks and activities, and capturing the lower levels of a capability's behavior is beyond the scope of KYKLOS. In this way, Behavior element was implemented as an abstract class, which means that it is not usable in the tool. Only the specializations are visible and usable by the users.

- The Intention element, which is another of the meta-elements of the meta-model, has been complemented with three specialization classes, to improve the tool's descriptive capability. The specializations are:
  ○ Goal ○ Problem ○ Requirement
  In addition, the specializations allow to capture the "purpose" attribute of the previously existing Capability State element, via their direct association to a capability. The Capability State captured what is the purpose of a capability, in terms of achieving a goal, avoiding a problem, or meeting a requirement, and if it actually succeeded in the fulfillment of the Intention element.

– Generalization of elements
  - Component was introduced; Process, as a Behavior element, and Resource, are both components of the Configuration class. This fact allowed the introduction of the Component abstract class, which is not visible and usable in the tool, but is the parent of both Component types and also gains their common Owner attribute.

– Utility addition
  - Resource pool, is a class that has no direct association to the phenomenon of capability change, however, its utility lies in the fact that the configuration components have been designed in a way that does not allow them to exist independently of a container. For this reason, the Resource pool element acts as a repository for the entire set of organizational resources that have not been allocated to a capability's configuration and improve partitioning potentials of a model.

Final implemented language meta-model.

The outcome of applying these changes to the initial meta-model is depicted in Fig.3, while the complete set of language concepts and their definitions are shown in Table 1.

Table 1. The complete set of language concepts and their definitions, from [5].

Concept	Description
Capability	A set of resources and behaviors, whose configuration bears the ability and capacity to enable the potential to create value by fulfilling a goal within a context.
Configuration	The set of resources that comprise the capability along with the behavior elements that deliver it. A capability may have several different configurations but only one may be active at any given moment in time.
Resource	Any human, infrastructure, knowledge, equipment, financial or reputation asset that can be used by an organization to enable the capability's realization. It can be allocated to one or more capability configurations, based on its capacity.
Resource pool	The complete set of an organization's available resources.

Context	All the factors that form the setting in which a capability exists, are relevant to its performance and within which the capability is perceived.
Outcome	The result of the capability's realization. Comparing it to KPIs and Intention elements can provide insight on whether a capability change is necessary or not.
KPI	A preset measurable value that expresses an important aspect of the context that a capability depends on to reach the desired outcome. Used to assess the efficiency of the capability's realization when compared with outcome values.
Monitored Factor	A context factor that has been identified and associated to a capability's performance and is being observed in relation to the capability. It is usually expressed as a KPI.
Intention element	An abstract element that includes all the concepts that refer to the intentions governing the capability, for example, goals, problems or requirements.
Goal	A desirable state that an organization aims to achieve. It is a type of Intention element.
Problem	An undesirable condition that an organization aims to avoid or tackle. It is a type of Intention element.
Requirement	A necessary state that an organization has to fulfill. It is a type of Intention element.
Behavior element	An abstract element that describes a structured set of activities whose execution delivers the value of the capability, for example, a process, service, activity or task.
Process	A behavior element that consists of activities aiming to fulfill a certain goal.
Change	Change represents the transition from one configuration to another. It can be described using several change properties. A capability change is finalized when a configuration's activity state is modified.

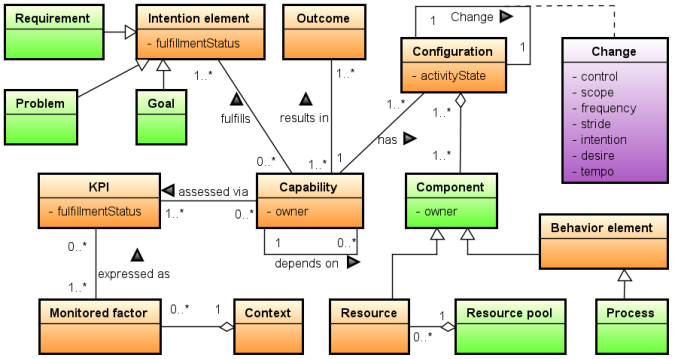


Fig. 3. The language meta-model, with the remaining (orange), converted (purple) and new (light green) classes.

4.2 Graphical Notation

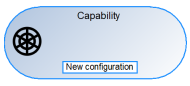
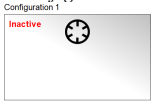
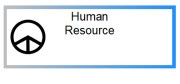
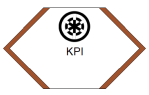
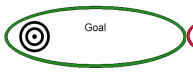
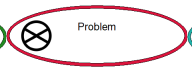
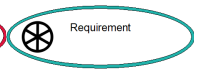
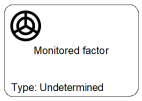
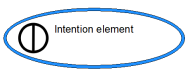
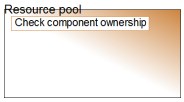
As mentioned earlier, an essential part of a modeling language is its notation. While the initial meta-model has been created using UML, for KYKLOS we introduced a new notation that combines both symbols and shapes. Symbols and shapes comprise the primary and secondary notation respectively. For the primary notation every concept of the language meta-model has been assigned a unique symbol, as shown in Table 2.

Table 2. The primary notation of KYKLOS.

Capability	*Configuration*	*Resource*	*Outcome*	*KPI*
				
Goal	*Problem*	*Requirement*	*Process*	*Change*
				
Monitored Factor	*Intention element*	*Resource pool*	*Context*	*Behavior element*
		Container	Container	N/A

Effort has been put to ensure the notation's short learning curve. This is achieved by a symbol set consisting of items that are consistent in terms of size, visual appearance and maximized simplicity, while in parallel preserving a clear distinction among them. The symbol color is black, to facilitate users with color deficiencies [19]. The secondary notation includes colored shapes but relying on color alone to distinguish image content is ineffective. The black symbols ensure that potential problems regarding compatibility with monochrome displays are avoided. Using color is not only for coding information but also for aiding visual search as the items become easily discriminable [19]. The secondary notation consists of standard shapes, i.e. polygons, ellipses and rectangles, and a set of colors that remain discriminable if superimposed on one another or juxtaposed [19], to improve memorability. The secondary notation includes the primary one and is complemented with text. Minimum elements have been used in both notations to avoid cluttered KYKLOS models, which has been a problem in earlier applications using the UML notation [9]. Table 3 depicts the secondary notation.

Table 3. The secondary notation of KYKLOS.

Capability	*Configuration*	*Resource*	*Outcome*	*KPI*
				
Goal	*Problem*	*Requirement*	*Process*	*Change*
				
Monitored Factor	*Intention element*	*Resource pool*	*Context*	*Behavior element*
				N/A

4.3 User Interaction

The last part of the implementation consists of technical additions provided using the ADOxx platform. These additions are implemented using the AdoScript language and provide automation that facilitates modeling in the tool and improves empirical quality of the model in terms of graph aesthetics [20]. Fig. 4 depicts these functionalities in an example KYKLOS model.

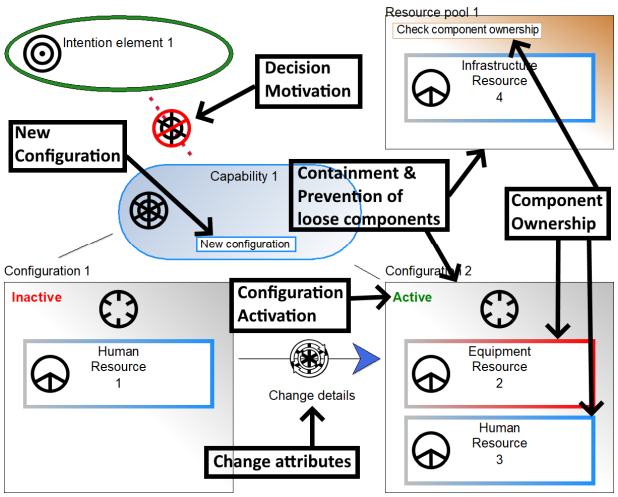


Fig. 4. User interaction facilities in the KYKLOS tool.

New configuration. Creating a new configuration is facilitated by a button existing on Capability objects. The tool creates and automatically connects a new Configuration, taking into consideration the spatial alignment of the object for increased visual quality.
Containment: This functionality uses the core ADOxx relationship "Is Inside". Having a model element graphically put inside another allows them to be related in an invisible way (no connecting lines required), which improves the model in terms of complexity by reducing crossing lines.
Configuration activation. Whether a configuration is active depends on whether the required components are allocated to it. In the tool, the required components are listed in a "Notebook" area. The tool checks on this list and calculates whether the components that are contained in a configuration match the list or not and activate or deactivate the configuration accordingly.
Component ownership. The ownership attribute captures if a component is owned by the same organization as the capability (internally) or not (externally). A button existing on Resource pool objects automatically calculates the ownership type and, changes the visualization of the component's right side border to blue (internal ownership) or red (external ownership) for improved comprehensibility. Similarly, it calculates and reports the externally owned components for consideration of organizational boundaries.
Prevention of loose components. Resources are components, so, they are not supposed to exist outside a container. For this reason, the tool does not allow the creation or movement of components if they are not contained.

Change attributes. An association exists between configurations that includes visually the attributes of change. This association class change includes a button that shows or hides the attributes of change in order to avoid clutter and complexity in larger models.
Decision motivation. KPIs and all Intention elements are connected to Capability with a special association called Status that is a control graphic element. Dependent on whether the object's content is fulfilled or not by the given capability, the visualization changes to facilitate identifying a reason for change, e.g. an unfulfilled goal.
Relationship grouping. Towards avoiding a large number of different association types, as in the language meta-model, all the associations except Status and Transition/Change are using the same visualization. However, strict rules have been coded to prevent using wrong association types in a produced model. This mitigates the risk of mistakes.

5 Discussion and Lessons Learned

The greatest opportunity addressed during the implementation was the potential to refine the initial meta-model into a simplified version in the language meta-model. This does not imply the loss or reduction of the initial meta-model's effectiveness. On the contrary, the KYKLOS language meta-model was expected to provide equal effectiveness with the initial meta-model, while in parallel avoiding the complexity and clutter that characterized the models produced using the initial meta-model, as in [9].

The most striking part of the meta-model transformation is the *reduction of the classes*, from 30 to 16, which indicates a significant simplification. In practice, a modeling tool that would have provided 30 available classes to a user would require a longer learning curve and modeling experience. The number of relationships has also been significantly reduced. Six associations share a common visualization that depicts the relationship status of the objects without a need to require additional learning steps from the user. Our lesson here is that the initial meta-model was created to cover "all" concepts of capability change and thus was not suited for creating a modeling language.

All the implementation activities bear their own advantages and disadvantages, often achieving a *balance between simplicity/utility and descriptive power*. Every intervention has been driven by advantages preponderating disadvantages. Introducing new classes increases the language meta-model's degree of complexity. However, all the introduced classes have provided either improved user experience, as for example, with the Resource pool class, or specified the more abstract concepts of the initial meta-model, as for example the Process and Goal classes specifying Behavior element and Intention element respectively. Similarly, it has been ensured that the removed classes have a minimal cost on the tool's descriptive power, for example, removing the Function class heavily simplified the model, and if desired, Functions can be described by other means such as creating separate models for each function. In both cases, we conclude that preponderance of simplicity or descriptive power has been the driver.

During the implementation it became clear that *the tool is more than the language meta-model*. The tool allows for more than just adding static concepts to a model, since it is possible to add functionalities too. For example, even if converting the Owner class to an attribute of two separate classes is considered a bad modeling practice in UML,

adding the component ownership functionality enables the tool to compare attributes of different classes to see if they are "owned" by the same organization. Converting classes to functionalities like this does not reduce descriptive potential, but it improves the user's interactivity combined with reduced complexity, making the change worthwhile.

The greatest challenge has been to *retain an operational and semantic consistency* between the initial and the language meta-model. The tool also needed to be operationally aligned with the modeling procedure, that is, to provide an adequate set of primitives for capturing the required elements for documenting, analyzing and communicating the phenomenon of capability change during the different KYKLOS method phases. This has been theoretically addressed, yet, a practical evaluation of the implementation by the actual users is required. KYKLOS is meant to be used both by technical and business people, therefore, the implementation needs to be evaluated both by users with modeling experience and those without any, a step which is already planned as a future step of the project.

Implementing the method with a variety of functionalities can *facilitate the user following a modeling process*. The dynamic automated aspects of the KYKLOS tool make steps towards an evolved version of modeling software that can guide the user's actions, as for example with the automatic capability configuration design, and mitigate the risk of syntactic mistakes, as for example with the restrictions applied on the association selection in the KYKLOS tool. These functionalities have been possible because of the ADOxx environment whose core platform enables different levels of automation.

Regarding ADOxx as the selected platform for the implementation of the KYKLOS method, its advantages as a specialized meta-modeling platform can be summarized in the pre-existing functions and meta-modeling structure that saves a significant amount of time and effort for the developer. In theory, taking into consideration that the tool's requirements are not platform-dependent, platforms like Eclipse are equivalent, however, in practice, ADOxx's existing functions are valuable, especially when it concerns cases where a concept needs to be converted to a tool functionality and dynamic behavior is required, as encountered in the KYKLOS implementation.

We aspire that the reported remarks can also benefit any implementation initiative that encounters similar opportunities and challenges, especially when the addressed phenomenon is as complex and dynamic as capability change.

6 Conclusions

In this paper, the implementation of the KYKLOS modeling method, specifically designed for the phenomenon of capability change, has been reported along with the lessons learned from the procedure. The initial meta-model has been adjusted and simplified to improve the resulting tool models in terms of complexity and clutter. Converting the initial meta-model's classes to attributes, association classes and ADOxx functionalities, along with the removal and introduction of classes led to the language meta-model, which has been complemented with a graphical notation and additional UI functions that aim to facilitate the user's overall experience of the KYKLOS method, in terms of applicability, learning curve and operational alignment with the tool.

References

1. Persson, A., Stirna, J.: An Explorative Study into the Influence of Business Goals on the Practical Use of Enterprise Modelling Methods and Tools. In: New Perspectives on Information Systems Development. pp. 275–287. Springer US, Boston, MA (2002).
2. Karagiannis, D., Kühn, H.: Metamodelling Platforms. In: Bauknecht, K., Tjoa, A.M., and Quirchmayr, G. (eds.) E-Commerce and Web Technologies. pp. 182–182. Springer (2002).
3. Fayoumi, A., Loucopoulos, P.: Conceptual modeling for the design of intelligent and emergent information systems. Expert Systems with Applications. 59, 174–194 (2016).
4. Fill, H.-G., Karagiannis, D.: On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform. EMISA. 8, 4–25 (2013).
5. Koutsopoulos, G., Henkel, M., Stirna, J.: Modeling the Phenomenon of Capability Change: the KYKLOS Method. In: (To appear in) Domain-Specific Conceptual Modeling: Concepts, Methods and Tools Vol.II. Springer International Publishing, Cham (2021).
6. Koutsopoulos, G., Henkel, M., Stirna, J.: An analysis of capability meta-models for expressing dynamic business transformation. Softw Syst Model. 20, 147–174 (2021).
7. Koutsopoulos, G., Henkel, M., Stirna, J.: Modeling the Dichotomies of Organizational Change: a State-based Capability Typology. In: Feltus, C., Johannesson, P., and Proper, H.A. (eds.) Proceedings of the PoEM 2019 Forum. pp. 26–39. CEUR-WS.org, Luxembourg (2020).
8. Koutsopoulos, G., Henkel, M., Stirna, J.: Requirements for Observing, Deciding, and Delivering Capability Change. In: Gordijn, J., Guédria, W., and Proper, H.A. (eds.) The Practice of Enterprise Modeling. pp. 20–35. Springer International Publishing, Cham (2019).
9. Koutsopoulos, G., Henkel, M., Stirna, J.: Conceptualizing Capability Change. In: Nurcan, S., Reinhartz-Berger, I., Soffer, P., and Zdravkovic, J. (eds.) Enterprise, Business-Process and Information Systems Modeling. pp. 269–283. Springer, Cham (2020).
10. Koutsopoulos, G., Henkel, M., Stirna, J.: Improvements on Capability Modeling by Implementing Expert Knowledge About Organizational Change. In: Grabis, J. and Bork, D. (eds.) The Practice of Enterprise Modeling. pp. 171–185. Springer, Cham (2020).
11. Guizzardi, G.: Ontological foundations for structural conceptual models, (2005).
12. Object Management Group (OMG): OMG® Unified Modeling Language®, https://www.omg.org/spec/UML/2.5.1/PDF, (2017).
13. Karagiannis, D., Mayr, H.C., Mylopoulos, J. eds: Domain-Specific Conceptual Modeling: Concepts, Methods and Tools. Springer, Cham (2016).
14. Koutsopoulos, G.: Managing Capability Change in Organizations: Foundations for a Modeling Approach, http://urn.kb.se/resolve?urn=urn:nbn:se:su:diva-185231, (2020).
15. Sandkuhl, K., Stirna, J. eds: Capability Management in Digital Enterprises. Springer (2018).
16. Ulrich, W., Rosen, M.: The Business Capability Map: The"Rosetta stone" of Business/IT Alignment. Cutter Consortium, Enterprise Architecture. 14, (2011).
17. OMiLAB: The ADOxx Metamodelling Platform, https://www.adoxx.org/live/home
18. Tsai, C.H., Zdravkovic, J., Stirna, J.: Capability Management of Digital Business Ecosystems – A Case of Resilience Modeling in the Healthcare Domain. In: Herbaut, N. and La Rosa, M. (eds.) AISE. pp. 126–137. Springer International Publishing, Cham (2020).
19. Post, D.L.: Color and Human-Computer Interaction. In: Handbook of Human-Computer Interaction. pp. 573–615. Elsevier (1997).
20. Krogstie, J.: Quality of Conceptual Models in Model Driven Software Engineering. In: Cabot, J., Gómez, C., Pastor, O., Sancho, M.R., and Teniente, E. (eds.) Conceptual Modeling Perspectives. pp. 185–198. Springer International Publishing, Cham (2017).