Development of Conceptual Models and Realization of Modelling Tools within the ADOxx Meta-modelling Environment

A Living Paper by the OMiLAB Community

Abstract: *A community for a specific domain forms around common, but specific artefacts found in that domain. For the domain of conceptual modelling the ADOxx meta-modelling environment is such an artefact, having formed a community through projects, realized tools and knowledge exchange. This paper presents the development of conceptual models and their realization in form of modelling tools using the ADOxx meta-modelling environment. First, it describes relevant concepts of conceptual models, the modelling layers and modelling methods before introducing their formalization using the ADOxx environment. Second, additional implementation cases are depicted with examples, providing references and elaborating further details on the realization of conceptual models. This contribution is built as a so-called "living paper" meant to be further extended in the future by members of the OMiLAB community.*

Keywords: *Conceptual Models, Conceptual Modelling, Model Value, ADOxx, Meta-Modelling, Modelling Tools*

1 INTRODUCTION

Research and development in a scientific field require sustainability of results and achievements for its community. For the field of computer science and more specifically information systems such sustainability can be achieved among other things through the software the community has formed around. Such software can be used to observe, evaluate and transport conserved knowledge between members of the community; it becomes a central part in working, discussing and sharing of knowledge between participants creating and (re-)using parts of the software.

The observation that software artefacts for a specific domain form the community is underlined by prominent examples such as the Eclipse community for software development, the Mozilla community for web development or the recently evolved Visual Studio Code community for advanced code editing. As such, the ADOxx community acts in the domain of conceptual modelling utilizing the environment of the ADOxx platform as a common denominator.

Throughout several years the ADOxx platform has been steadily employed in several European, national and individual research projects (CloudSocket[1], complAI[2], DISRUPT[3] and more[4]), used to create multiple tools[5] and as an integral part of the knowledge exchange on conceptual models with its community members. Educational activities have promoted its use, e.g., in the international NEMO Summer School[6] series.

This paper is a result of the OMiLAB community with a specific version at the current date of publishing. However, we aim to support its further growth and extension with additional content, materials etc., applying

[1] CloudSocket webpage: https://site.cloudsocket.eu/ (accessed 2021-06).
[2] complAI webpage: https://complai.innovation-laboratory.org/ (accessed 2021-06).
[3] DISRUPT webpage: http://www.disrupt-project.eu/ (accessed 2021-06).
[4] More projects using ADOxx can be found at https://www.adoxx.org/live/development-spaces (accessed 2021-06).
[5] Overview of modelling tools available at https://projects.omilab.org (accessed 2021-06).
[6] See https://nemo.omilab.org/ (accessed 2021-06) for more on the NEMO Summer School.

an approach similar to open-source software by creating a living paper. Every member of the OMiLAB community is invited to participate in enhancing and extending this text with own results and achievements. For further information and the latest version visit https://www.omilab.org/activities/living-paper/develop+realize-within-adoxx/ (accessed 2022-03).

In the remainder of the paper, we will first introduce the essentials of conceptual modelling through the concepts used to both develop such models as well as realize the corresponding modelling tools in section 2. Section 3 shows general application cases of models in an ADOxx based modelling tool with examples, distinguishing different types of interactions. Section 4 wraps up the paper with a conclusion.

2 ESSENTIALS

When developing conceptual models, it is important to consider the domain they are created for, what purpose they are for and on what level in the modelling hierarchy they are positioned. Relevant levels of that hierarchy can be seen in Figure 1. It also shows the relation between the levels and two parts of the ADOxx platform: the development tools and the modelling tools. On the model level the main concern is depicting a system under study (the "original"), which can be either something that already exists (as-is as a descriptive model) or something that should exist (to-be as a prescriptive model). To be able to create models a modelling language is required. The modelling language in turn can be depicted by a meta-model that is realized in a modelling tool[7]. These meta-models are developed beforehand with a specific language, in this case a meta-modelling language. Again, the meta-modelling language can be depicted through another model: the meta²-model (meta-meta-model) which is realized in the development tools. The hierarchical approach assumes a language-oriented understanding of the models as introduced in [1] and further refined by [2].

While this approach could be further applied to allow customizable and adaptable meta^x-models, it must end at some point for practical reasons to create a runnable application. In case of the ADOxx platform the highest implemented level is the meta²-model, which uses a meta²-modelling language that can be executed on a computer (a programming language). Thus, it provides a meta-modelling language that can be used to create own meta-models that are then realized as modelling tools.

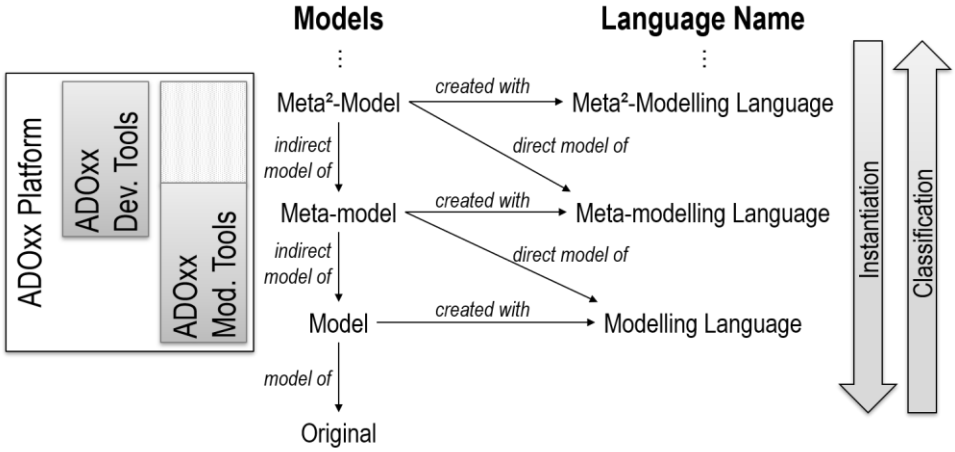

Figure 1: Modelling hierarchy and language levels (adapted from [2]).

[7] While the focus of the ADOxx modelling tools is on the meta-model and model levels, they are influenced by the concepts existing on the meta²-model level.

When developing and realizing your own modelling language with the ADOxx platform the purpose is generally to create a modelling tool which can be used for the creation of models with your modelling language, thus the domain of conceptual modelling itself becomes relevant. In conceptual modelling there are three important concepts to specify a modelling language as depicted in Figure 2: Concept, Characteristic and Connector.

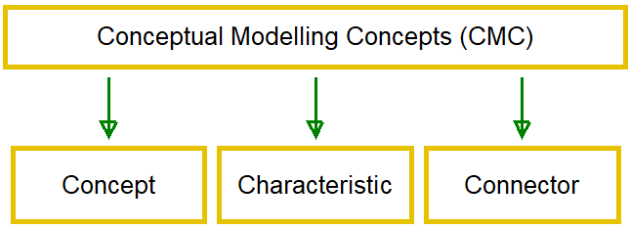

Figure 2: Conceptual Modelling Concepts.

The three Conceptual Modelling Concepts deal mainly with the structure of the modelling language by describing its elements. **Concepts** represent something that is relevant and worth to be modelled. **Characteristics** don't provide any substantial value by themselves, but instead are used to describe other things, providing relevant details. **Connectors** link or connect things, introducing a relation between them. The declaration of Concepts, Characteristics and Connectors provides a conceptual blueprint of elements within the modelling language and consequently what can be used in models adhering to it.

According to [2] it is insufficient to only consider the general structure of a modelling language. Further aspects of the elements also need consideration: their semantics (meaning), their notation (visual representation) and their syntax (rules for creating models). Moreover, modelling languages by themselves are not the only thing of interest, but also in relation to other concepts as shown in Figure 3.

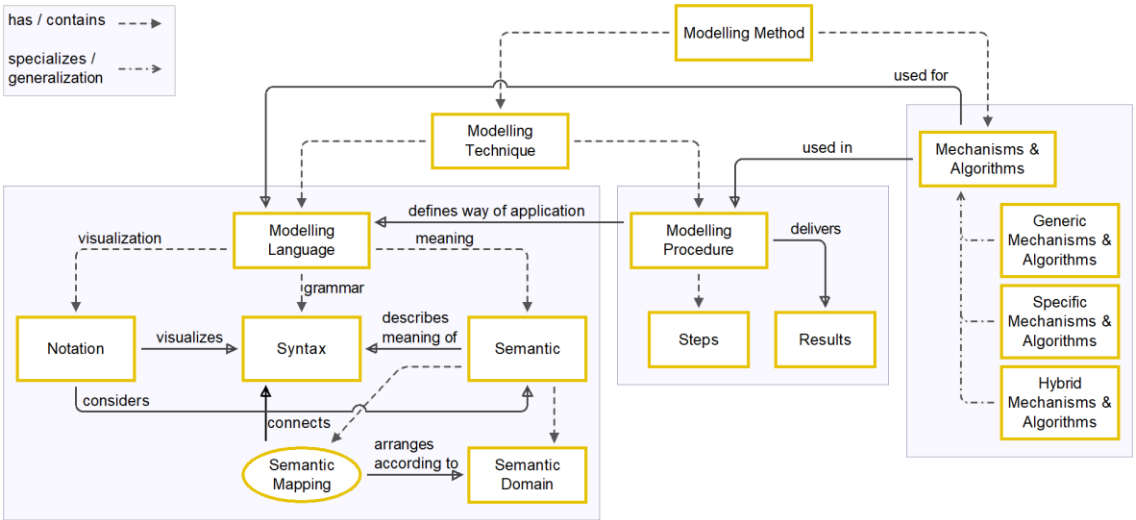

Figure 3: Modelling language, technique and method (based on [2]).

Besides the modelling language itself there is also a procedure on how the language is intended to be applied to achieve specific results, where the restrictiveness of the procedure can vary. Additionally, mechanisms and algorithms support the procedure in achieving those results by processing the available model data. All these parts together form the modelling method. These parts need to be considered together, since they influence each other: a step of the procedure could be enhanced for example by a mechanism, which in turn could require certain elements in the modelling language to be present or certain combinations of elements to be prohibited in a model.

We like to also mention that there is a process for the formalization on three different layers, although this is not the core issue of this paper: definition, specification and implementation (see also Figure 4, [3] and [4]).

A language which supports among other things the description of a modelling language, its purpose, procedure and functionalities, allowing to capture its requirements has been discussed in [5]. It is called CoChaCo, which builds on the ideas of Concept, Characteristic and Connector and extends it with additional elements to allow the description of modelling methods[8].

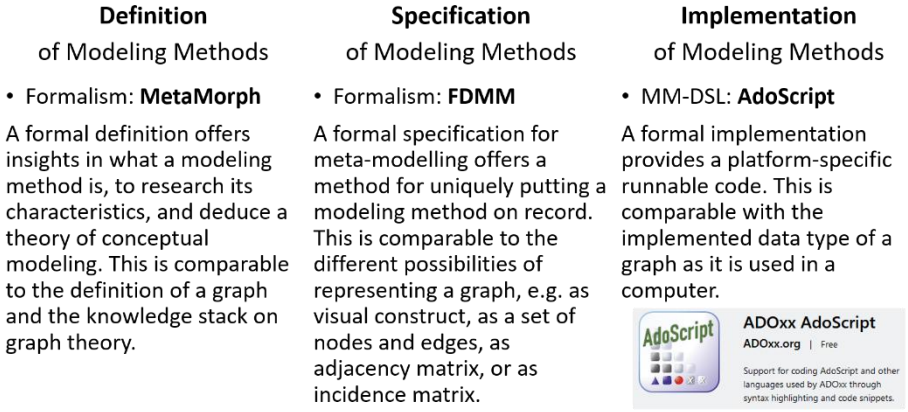

Figure 4: Modeling methods - different layers of formalization.

The CoChaCo language has also been used to describe the ADOxx platform, providing an overview of the concepts and functionalities (see Figure 5).

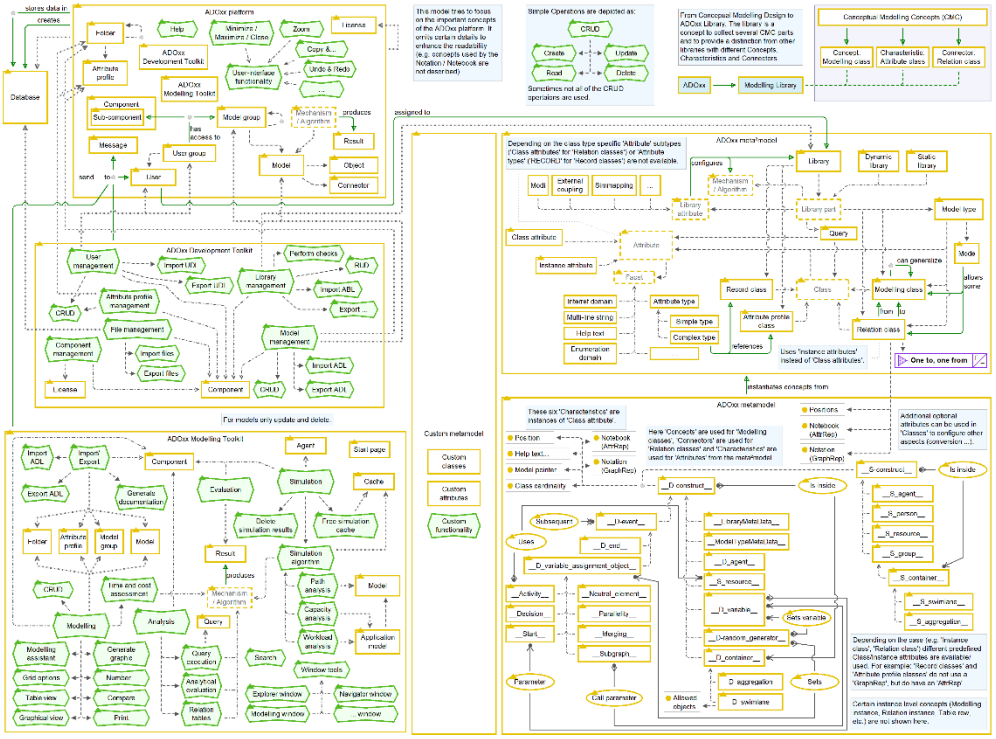

Figure 5: Overview of the ADOxx platform components, functionalities and meta(²)-model concepts[9].

[8] An implementation is available at https://www.omilab.org/cochaco (accessed 2021-05).

[9] A more readable version can be found at https://www.omilab.org/adoxx/ (accessed 2021-09).

The left side describes the platform's major components and functionalities, like the creation and management of models, the execution of queries and the import / export into specific format on model level, but also the creation and management of modelling method implementations through so called "libraries" and file management on meta-model level.

The right side of the image contains a description of the structural aspects of the ADOxx platform, describing its meta²-model and a meta-model that is provided as a starting point. These are used to realize the necessary parts of a modelling language through instantiating the main concepts: modelling classes, relation classes and attributes. For these the syntax, notation and semantics can be further described, through the relationships between these concepts (syntax), through assignment of visualizations (notation) or through help texts for humans and code for the machine (semantic).

Mechanisms and algorithms can be implemented through code and activated through the platform. Pre-packaged and configurable functionalities provided by the platform can also be adapted and used. The procedure can be supported through the mechanisms and tailored functionalities can guide the user through the different steps[10]. The tall element in the middle of Figure 5 represents the own meta-model to be realized and the configuration of the other elements of the ADOxx platform in order to provide a modelling tool for the desired modelling method.

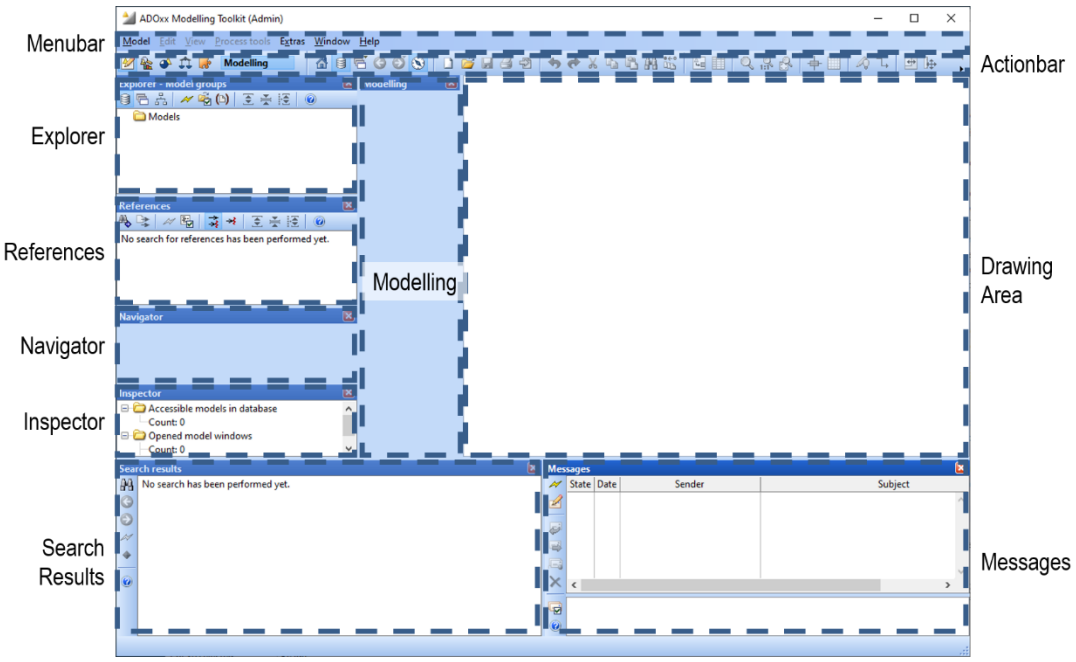

Figure 6: Parts of the user-interface of an ADOxx based modelling tool.

Depending on the realization of one's own modelling method, different parts of the ADOxx platform's user interface are relevant in the resulting modelling tool. Figure 6 shows some of the different components that are available in the user interface of an ADOxx modelling tool. Some of these are generally the same in every implementation, like the Explorer showing the available models structured in folders. Some dynamically change based on the meta-model, like the Modelling bar showing the types of elements available for an

[10] Note that these are just some of the elements of the ADOxx platform used to realize a modelling method. There are other elements to handle additional details like the representation of attributes, cardinalities or elements of the user interface.

opened model. Others have a default configuration that can be further customized, like adding new entries to the Menubar or the Actionbar.

An often-applied scenario for developing a modelling method and realizing a modelling tool with the ADOxx platform consists of using the following tools related to Figure 7:

1. CoChaCo[11] to design the modelling method by capturing the requirements for its elements, their allowed structure, the supported procedures and the relevant functionalities.
2. The ADOxx Development Toolkit[12] to instantiate the elements of the modelling language through modelling classes, relation classes and attributes. This can be achieved through the user interface or by directly writing ADOxx Library Language (ALL) code. This creates a configuration of the modelling language (the "library") that is interpreted by the ADOxx Modelling Toolkit.
3. Visual Studio Code with the ADOxx AdoScript extension[13] to write code for the configuration and the desired functionalities, e.g., using AdoScript to implement the functionalities or using GraphRep code to specify the notation.

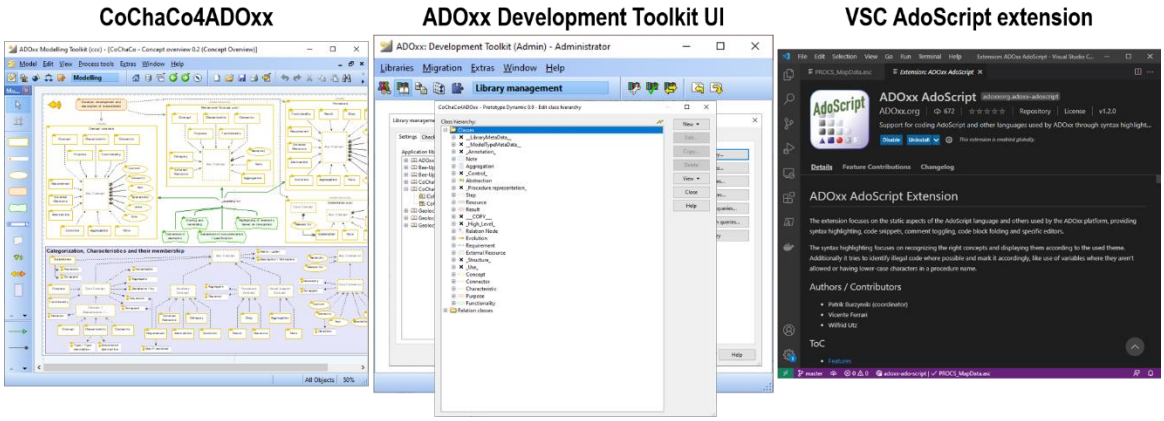

Figure 7: Three tools used for the development and realization of modelling methods.

Depending on the requirements of the modelling method different issues have to be considered for the implementation. The next section introduces some approaches for different kinds of requirements. For example, if simulation of models is required, then an approach can be found in the section "3.3 Simulation of Process Models".

For additional details on the conceptualization of modelling methods with the ADOxx platform see [6] and [7]. Some of the modelling tools realized with the ADOxx platform can also be found on the OMiLAB webpage[14].

[11] Available from https://www.omilab.org/cochaco (accessed 2021-05).
[12] Available from https://www.adoxx.org (accessed 2021-05).
[13] Available from https://marketplace.visualstudio.com/items?itemName=ADOxxorg.adoxx-adoscript (accessed 2021-05) or in the VSC by searching for the extension "ADOxx AdoScript".
[14] Overview of modelling tools available at https://projects.omilab.org (accessed 2021-06).

3 APPLICATION CASES

The ADOxx platform supports different types of interactions with actors or agents for modelling and development of modelling methods. For one its modelling capabilities and models can be used for both the interaction with (a) human users as well as with (b) other systems, like other applications or web-services. Additionally, the interactions can be motivated either to create models or to utilize models (e.g., execute, simulate alternatives, etc.), providing value to the user. The realization of these interactions can be done by the ADOxx platform itself, using external components like scripts, dynamically linked libraries and web-services or a mix of both. Implementing a specific interaction using exclusively functionality from the platform requires that all the relevant information can be found in the meta²-model, the meta-model and the model themselves.

Figure 8: Types of considered interactions between the ADOxx platform, users, other systems and models.

In this section we are showing some cases with different variations on the kinds of interactions depicted in Figure 8 that go beyond the basic creation and use of models for communication between humans.

3.1 DYNAMIC, INTERACTIVE NOTATION

One possible interaction with the user is through dynamic notations, which change based on the current state of the model. This can go beyond the simple depiction of values entered by the user; additionally, performing calculations and adding shapes and icons based on their results, showing or hiding specific details or depicting external graphics. Furthermore, the notations can provide interactable points which allow the user to access functionalities directly through the drawing area. These kinds of functionalities are generally implemented for models of a specific modelling language and can help both creating models (e.g., by inserting new objects) and using models (e.g., by starting a simulation).

3.1.1 Petri Net Transitions with "Fire" Buttons

An example for this can be found in the realization of the Petri Net language in the Bee-Up modelling tool [8][15]. The implementation of the transitions dynamically evaluates whether they are ready to fire according to the execution semantics of Petri Nets and if so, then a "Fire" indicator is visualized. The "Fire" indicator also serves as a button that can be clicked on, which fires the transition in the model and moves any tokens according to the specified net.

Both the dynamic notation and the result of using the interactive "Fire" button can be seen in Figure 9. The model depicts a simple Petri Net of available molecules and their chemical reactions. The state of the net changes after certain transitions fire and rearrange the tokens, influencing the readiness of the transitions and thus the available "Fire" buttons.

[15] An implementation is available at https://bee-up.omilab.org/ (accessed 2021-05).

While the example shows how such functionality can support the building of models, the implementation in Bee-Up also provides means to further use the models. It allows to specify so called “Effects” for transitions through AdoScript code. These are triggered when the transition fires, allowing to use all the processing and interaction capabilities of the platform and further extend the interaction between the user, the model and possibly other systems.

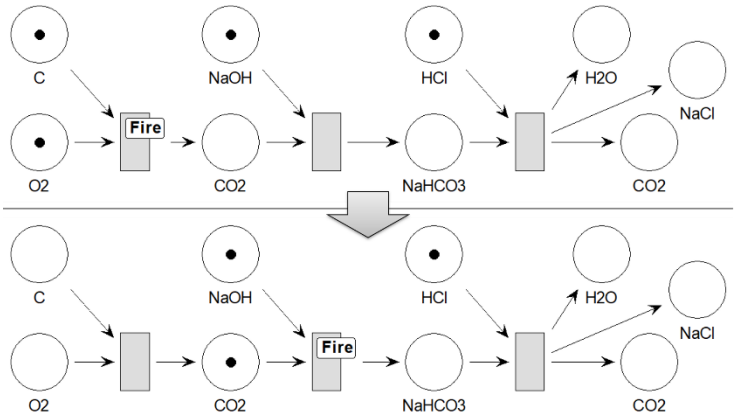

Figure 9: Example for dynamic and interactive notation in Bee-Up Petri Nets (model based on [9]).

3.2 Conceptual Models in a Machine-Readable Format

Another form of interaction allows the use of models by providing them in a machine-readable format, based on established standards, file formats, etc. These can then be further processed by other applications or services. Depending on the desired level of specificity these functionalities can either be implemented for models of a specific modelling language or on the meta²-model level to be language independent. While the latter works for models of any language it also means not reflecting specific semantics that a modelling language provides to the model when transforming the model into the desired format.

3.2.1 Use of models in an RDF format with a BPMS

An approach where the content of ADOxx models is exported in an RDF (Resource Description Framework) format is described in [10]. The exposed data from the models is linked with other sources of data and further enhanced using relevant reasoning axioms/rules. All these together provide a more complete view of the relevant part of the world as a knowledge base, which is used by a Business Process Management System (BPMS) frontend. Therefore, the interaction is here mainly with other systems, targeting to provide value from created models.

Figure 10 depicts a case of how this can be used. The left side shows parts of two connected models about the tasks and actors for the transportation of materials and the RDF graph these parts are transformed into. The right side shows parts of external data about parking lot locations that have also been transformed into an RDF graph. Both graphs are linked together and are further extended with results from reasoning axioms and GeoSPARQL annotations, providing geospatial context. The image depicts the links and annotations through dashed arrows. This enhanced RDF graph provides the knowledge base used by a BPMS frontend, where relevant information is shown to a logged in user, like their active tasks and parking recommendations (not depicted).

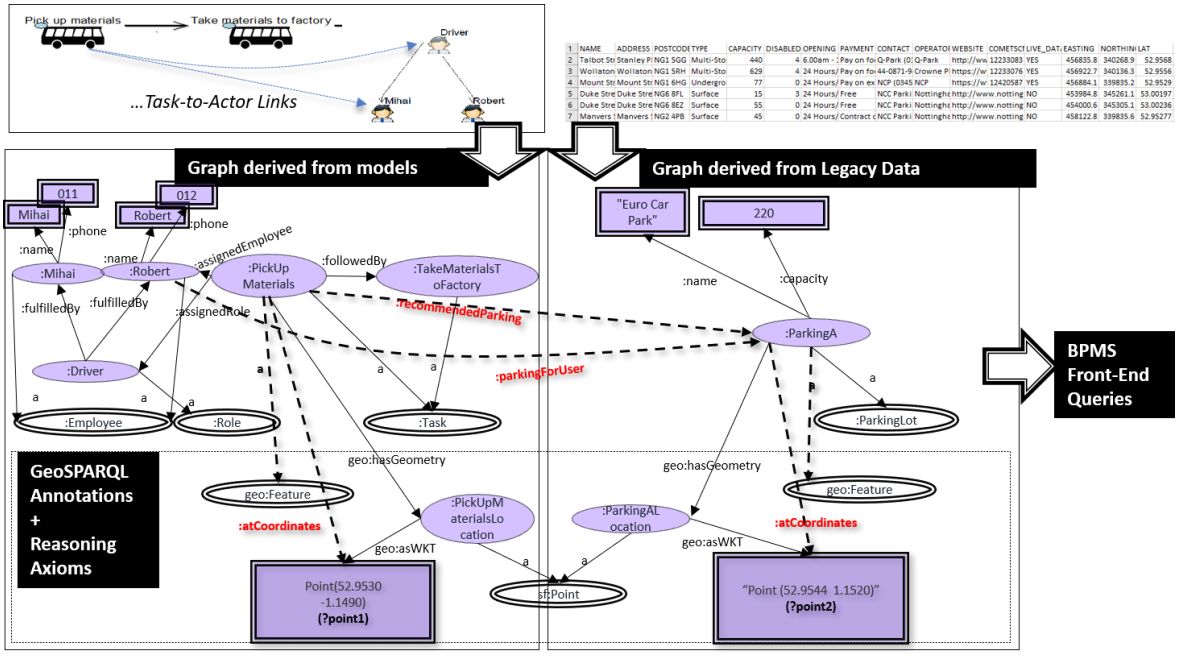

Figure 10: Example case using models transformed into RDF with other RDF data and reasoning rules for a BPMS (based on [10]).

A service enabling the transformation of ADOxx models to RDF is available on ADOxx.org[16]. It implements several transformation rules, both for models and meta-models. The rules themselves are realized on a meta-model independent level, relying on generic concepts like instances and relations linked to concepts of the meta-model. This allows the RDF transformation to be used for any meta-model and any model.

3.3 Simulation of Process Models

Implementations based on the ADOxx platform can also provide an interaction for creation and use of process models through simulation capabilities. They support the creation of models by allowing to compare the results from a simulation of an as-is model with data from the real world. Furthermore, to-be models can be created and compared with the as-is model through simulation. These kinds of functionalities may be implemented for a specific modelling language or on a more generic level of abstraction of the language, covering concepts of processes in general.

ADOxx provides a configurable implementation for process model simulation for a specific abstraction of process modelling languages. It allows to perform a path analysis of process models, as well as a capacity analysis or workload analysis when paired with models describing the organizational or working environment the processes are executed in. A path analysis simulates the process model multiple times to find possible paths through the process, as well as their quantitative metrics (e.g., execution time, cycle time, etc.) and probabilities based on the performed simulations. The capacity analysis runs a simulation of process models and determines results for them in a specific environment, like how much time a specific performer would spend doing certain work. The workload analysis goes further and not only combines the process models with their environment, but also considers waiting times, queues and a calendar when simulating the work of performers. Considering the oscillation in starting the process, as well as individual workload of performers this may lead to results showing unexpected or unhealthy accumulation of work.

[16] RDF transformation service available at https://www.adoxx.org/live/rdf_details (accessed 2021-05).

3.3.1 Simulation of BPMN Models

The Bee-Up modelling tool [8][17] configures the simulation capabilities provided by the platform to for the Business Process Model and Notation [11]. Together with an implementation for Working Environment Models it allows for all three types of simulations provided by ADOxx.

Figure 11 shows an exemplary process model in BPMN, describing the repair of a home appliance consisting of several tasks: communication with the customer, determining the fault, obtaining spare parts, performing the repair and disposing in case a repair is not performed, among others. It also contains a loop back if the repairs did not solve the problem. The path analysis has run 1000 simulations for the model and has found six different paths. The results for the two most likely paths are shown, with quantitative information like the expected execution time or cycle time for each path. The second path is also highlighted in the image.

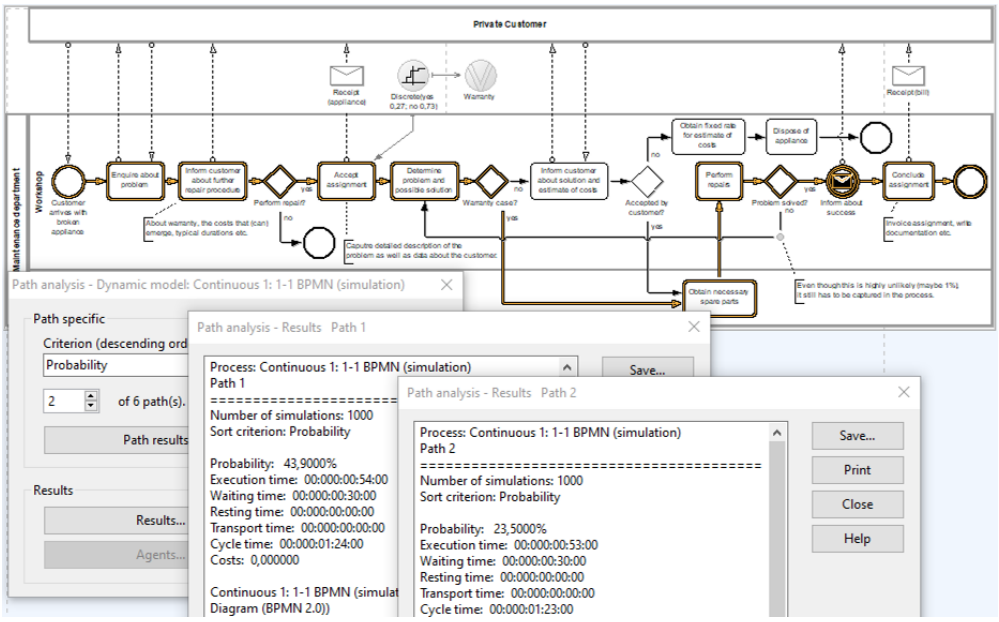

Figure 11: Example BPMN process with results from a path analysis in Bee-Up.

Assuming the model to represent the current state of the process, the results of the model simulation can be checked against reality. Based on those results the process can be adapted, by either changing certain parameters or altering the structure of the flow, creating alternatives. Those alternatives can then be simulated to compare with the original model.

3.4 Model Adaptation through Internet Resources

Interactions with resources found on the internet can be beneficial for the creation of models: providing data to either create new parts of the model or adapt existing parts. Cases where the data is used for other purposes are possible as well, like for the validation of models. A benefit of using resources from the internet is the possibility of retrieving the data at the time when it is needed and receiving a current version. The connection to the resources may be realized through simple HTTP requests. As such the interaction between the ADOxx platform generally happens with other webservers.

[17] An implementation is available at https://bee-up.omilab.org/ (accessed 2021-05).

3.4.1 Displaying a Map Image as Part of a Model

A case where an image is retrieved from the internet and used in a model is realized in the GeoLocation modelling implementation used as part of teaching ADOxx[18]. The implementation displays a map centered on a specific location as part of a model and allows placing additional elements on or around it. It works by providing a name for the location as an attribute of the model, which is then used to determine the latitude and longitude by calling a web-service[19]. The latitude and longitude are then used to retrieve an image of the map portion from the internet[20] and visualize it as the model background. Both services are available through simple HTTP calls, providing the result like a resource on the internet.

Figure 12 shows an example where an empty model is created, where "Vienna" is specified as the name of the location. Afterwards the two-step functionality is executed. The first step retrieves all locations that fit the name "Vienna". Since there are several options available (in Austria, France, United States, etc.) it prompts the user to specify which option they want to use. Having selected the first one in Austria provides the latitude and longitude for that location, which is then used to retrieve the map image and set it as the background of the model.

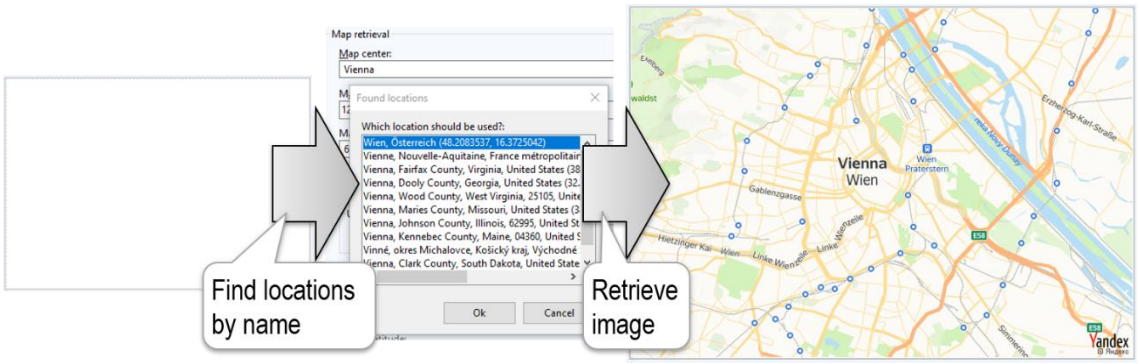

Figure 12: Example of retrieving the location and visualizing a map image for Vienna as the model background.

3.5 Integration with Cyber-Physical Systems

The ADOxx platform enables the integration with cyber-physical systems (CPS, e.g., robots), allowing additional capabilities for models when interacting with physical devices. Such models and CPS interactions can for example be used to control the behavior or to represent the current state of the CPS. While the main interaction is with other computer systems or agents, the models can also provide feedback to the user of the modelling environment. The realized integrations can be on varying levels of specificity in the modelling method: from a generic approach that allows to send any form of HTTP request to methods tied to interfaces of a specific CPS.

3.5.1 Controlling a Robot through Petri Nets

A simple integration with CPS is possible in the Bee-Up modelling tool [8][21] on a very generic level. As mentioned in the section "3.1.1 Petri Net Transitions with 'Fire' Buttons" a transition that fires can also trigger an "Effect". These effects are specified for each transition in the model using AdoScript code, which allows

[18] For example: the OMiLAB ADOxx Crash Course from March, details can be found at https://www.omilab.org/activities/events/adoxxcrashcourse2021_march/ (accessed 2021-05).

[19] Using the OpenStreetMap Nominatim service. For more details see https://wiki.openstreetmap.org/wiki/Nominatim (accessed 2021-05).

[20] Using the Yandex Static Maps API. For more details see https://yandex.com/dev/maps/ (accessed 2021-05).

[21] An implementation is available at https://bee-up.omilab.org/ (accessed 2021-05).

for a wide range of possibilities. One of them is to send HTTP requests [22] to an interface providing functionalities to control a robot. In this case the integration with CPS is not on the domain level of the modelling method, not in the semantics of the used modelling language elements, but instead realized as the implementation provides capabilities on a generic level.

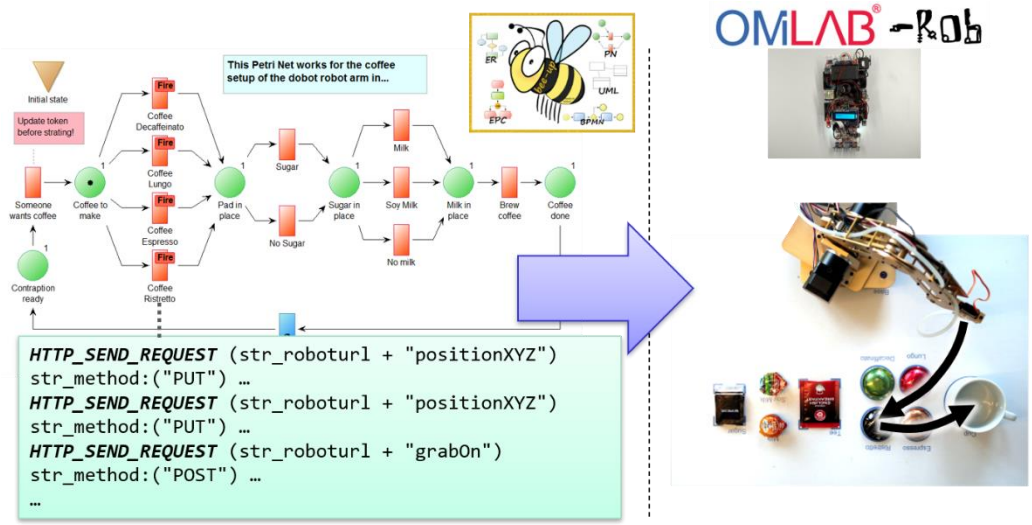

Figure 13: Example of a Petri Net in Bee-Up which controls behavior of a robot arm.

An example for a model which controls the behavior of a robot arm can be seen in Figure 13. The example[23] uses a scenario where a robot arm is used to collect the ingredients to make coffee. It consists of a Petri Net whose transitions contain the instructions to move a robot arm and the corresponding hardware which is part of OMiLAB[24]. The instructions entered as the "Effect" of a transition specify the requests that are sent to the HTTP interface provided by the robot arm. While the robot arm provides several simple instructions (move to position, grab on, let go), these are bundled together into more meaningful actions (use Ristretto coffee, add sugar, etc.) in the transitions.

4 CONCLUSION

Some basic concepts for developing conceptual modelling methods and realizing a modelling tool with ADOxx have been presented in this paper. Starting with the relevant levels of models (original, models, meta-models and meta²-models) and their relation to the ADOxx platform, through the basic concepts used in a modelling language (Concept, Characteristic, Connector) to the modelling method framework presenting additional aspects and elements that have to be considered besides a modelling language (modelling language, modelling procedure, mechanisms and algorithms). Based on this a general introduction on the realization using the ADOxx platform was given. It touched the concepts, components and tools employed to implement a modelling language, the necessary mechanisms & algorithms and to support the modelling procedure.

Afterwards the realization of different interactions between users, modelling tool and other systems have been presented, considering cases focusing on interaction with the user (e.g., section "3.1 Dynamic,

[22] Using for example https://www.adoxx.org/live/extended-http-requests-details (accessed 2021-05).

[23] The example scenario can be found on the Bee-Up page: https://bee-up.omilab.org/activities/bee-up/scenario-details/ (accessed 2021-05).

[24] See https://www.omilab.org/nodes/innovation.html (accessed 2021-05) for more information.

interactive notation") or on interaction with other systems (e.g., section "3.2 Conceptual models in a machine-readable format"). Additionally, the interactions can be motivated for the creation of models (e.g., section "3.4 Model Adaptation through Internet Resources"), the use of models to gain additional value (e.g., section "3.5 Integration with Cyber-Physical Systems") or both (e.g., section "3.3 Simulation of Process Models").

Every reader is also invited to join the OMiLAB community at www.omilab.org, develop and realize new modelling methods and participate in writing this living paper. Everybody interested in learning more about the details of realizing a modelling tool with the ADOxx platform is also invited to join one of the **freely available** ADOxx trainings[25].

REFERENCES

[1] S. Strahringer, "Ein sprachbasierter Metamodellbegriff und seine Verallgemeinerung durch das Konzept des Metaisierungsprinzips," in *Modellierung '98, Proceedings des GI-Workshops*, Münster, 1998.

[2] D. Karagiannis and H. Kühn, "Metamodelling Platforms," in *Proceedings of the Third International Conference EC-Web 2002*, Aix-en-Provence, 2002.

[3] V. Döller and D. Karagiannis, "Formalizing Conceptual Modeling Methods with MetaMorph," in *Enterprise, Business-Process and Information Systems Modeling. BPMDS 2021, EMMSAD 2021*, 2021.

[4] H.-G. Fill, T. Redmond and D. Karagiannis, "FDMM: A Formalism for Describing ADOxx Meta Models and Models," in *Proceedings of ICEIS 2012 - 14th International Conference on Enterprise Information Systems*, 2012.

[5] D. Karagiannis, P. Burzynski, W. Utz and R. A. Buchmann, "A Metamodeling Approach to Support the Engineering of Modeling Method Requirements," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*, Jeju, Korea (South), 2019.

[6] H.-G. Fill and D. Karagiannis, "On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform," *Enterprise Modelling and Information Systems Architectures - An International Journal,* vol. 8, no. 1, 2013.

[7] D. Karagiannis, "Agile Modeling Method Engineering," in *PCI '15: Proceedings of the 19th Panhellenic Conference on Informatics*, Athens, Greece, 2015.

[8] D. Karagiannis, R. A. Buchmann, P. Burzynski, U. Reimer and M. Walch, "Fundamental Conceptual Modeling Languages in OMiLAB," in *Domain-Specific Conceptual Modeling*, Springer International Publishing, 2016, p. 594.

[9] C. A. Petri and W. Reisig, "Petri net," 2008. [Online]. Available: http://www.scholarpedia.org/article/Petri_net. [Accessed 05 2021].

[10] M. Cinpoeru, A.-M. Ghiran, R. A. Buchmann and D. Karagiannis, "Model-Driven Context Configuration in Business Process Management Systems: An Approach Based on Knowledge Graphs," in *Perspectives in Business Informatics Research*, Katowice, Poland, 2019.

[11] OMG, "Business Process Model and Notation," 2014. [Online]. Available: https://www.omg.org/spec/BPMN/2.0.2/. [Accessed 05 2021].

[25] More details available at https://www.omilab.org/adoxx (accessed 2021-06).