# View-Based Near Real-Time Collaborative Modeling for Information Systems Engineering

Petru Nicolaescu[1(✉)], Mario Rosenstengel[1], Michael Derntl[2], Ralf Klamma[1], and Matthias Jarke[1,3]

[1] RWTH Aachen University, Lehrstuhl Informatik 5, Ahornstr. 55, 52074 Aachen, Germany
{nicolaescu,rosenst,klamma,jarke}@dbis.rwth-aachen.de
[2] Eberhard Karls Universität Tübingen, eScience-Center, Wilhelmstr. 32, 72074 Tübingen, Germany
michael.derntl@uni-tuebingen.de
[3] Fraunhofer FIT, Birlinghoven Castle, Sankt Augustin, Germany

**Abstract.** Conceptual modeling is a creative, social process that is driven by the views of involved stakeholders. However, few systems offer view-based conceptual modeling on the Web using lock-free synchronous collaborative editing mechanisms. Based on a (meta-)modeling framework that supports near real-time collaborative modeling and metamodeling in the Web browser, this paper proposes an exploratory approach for collaboratively defining views and viewpoints on conceptual models. Viewpoints are defined on the metamodeling layer and instantiated as views within a model editor instance. The approach was successfully used for various conceptual modeling languages and it is based on user requirements for model-based creation and generation of next-generation community applications. An end-user evaluation showed the usefulness, usability and limitations of view-based collaborative modeling. We expect that Web-based collaborative modeling powered by view extensions will pave the way for a new generation of collaboratively and socially engineered information systems.

**Keywords:** Views · Viewpoints · Collaborative conceptual modeling

## 1 Introduction

Conceptual modeling is a key tool for representing domain-specific information during the requirements elicitation and design phases of information systems [16]. With the increased collaboration between stakeholders from different geographical locations and the emergence of Web technologies that enable near real-time (NRT) communication and offer a proper medium for collaboration and information exchange, new research opportunities emerge in the field of collaborative conceptual modeling. Usually, (meta-)models are used to create abstract representations of a system and to address different groups of stakeholders,


S. Nurcan et al. (Eds.): CAiSE 2016, LNCS 9694, pp. 3–17, 2016.
DOI: 10.1007/978-3-319-39696-5_1

e.g. developers, project managers, customers, partners, or investors. The complexity of models and metamodels representing these systems is increasing rapidly. Thus, it is often necessary to look at a complex system or application from different points of view. Moreover, certain stakeholders may prefer or require a different view to express their concerns [11,17]. Views (also referred to as viewpoints) are used to deal with this complexity in (meta-)modeling frameworks. View-based modeling aims at creating partial metamodels and models, each one of them reflecting a set of concerns of the modeled system [17]. Previous research during the late '90 s covers viewpoints and the generation of views from metamodels, especially from a requirements engineering perspective [15,16]. The works also explore the generation of views from metamodels, in various industry or academia-driven information systems. However, as aforementioned, the Web 2.0 uprise has reshaped the social work behavior, making systems available for heterogeneous communities free or at a low cost, on the Web, which is also valid in the conceptual modeling case.

The motivation underlying this work is taken from a Model Driven Web Engineering scenario where developers and end users collaboratively built a Web application [4]. In this setting, a study was carried out using thirteen user evaluation sessions in groups of two or three, with a total of 36 participants. We observed how collaborative modeling can be leveraged by different stakeholders during the design phase of an information system. We provided both non-technical users and software developers with an NRT collaborative editor for modeling and generating Web applications. Based on a common application metamodel, they were given separate parts of a model for modeling the frontend and backend of the application. In a first stage, participants were allowed to model the backend of a given information system in NRT (services, database, service interface, etc.). Then, they were asked to model in the same way the interface for the already designed backend. The results show that many end users perceived parts of the model as too complicated or were finding the representation not relevant or intuitive for them. As such, they expressed the desire to reduce complexity of a model by only being presented with relevant aspects, which in the specific case were mostly considered to be HTML5 frontend elements. Furthermore, many end users expressed the desire to have familiar representations of such objects. Following these outcomes, this paper explores the NRT collaborative definition of views through a metamodel-based approach and their usage in NRT collaborative modeling scenarios.

We formulate three research questions:

- How can various stakeholders collaboratively define views as part of a metamodel in NRT? (*RQ1*);
- How to generate customized views based on the metamodel definition where stakeholders can further collaboratively edit parts of a generated model? (*RQ2*);
- Do views impact NRT collaborative modeling with respect to user experience improvement and the modeling process speed (*RQ3*)?

In order to explore the collaborative work with custom-defined views, the paper presents a view extension framework for NRT collaborative modeling in the Web browser. The framework facilitates a collaborative, graphical definition of views on the metamodel layer. This allows metamodelers to redefine entities (e.g. objects and relationships of a metamodel) in custom viewpoints and then apply these viewpoints to the models. In previous work [5], we presented SyncMeta, a Web-based NRT collaborative (meta-)modeling tool. SyncMeta allows the collaborative creation of metamodels in NRT based on a visual language specification (VLS) and the generation of model editors based on the defined metamodels. The view extension was implemented on top of this framework.

This paper is structured as follows. In the next section we introduce the SyncMeta framework that offers the foundation for our collaborative view-based modeling extension. Section 3 introduces viewpoints and views and provides a formalization on view-based metamodeling. Section 4 then describes the architecture and implementation of the framework and discusses the limitations of our approach. Section 5 presents an end-user evaluation of the implementation. Section 6 shows how the view-based extension goes beyond the state of the art. Finally, Sect. 7 concludes the paper and outlines the future work.

## 2 SyncMeta: Near Real-Time Collaborative (Meta-)Modeling and the Views Extension

SyncMeta is a Web-based metamodeling framework that allows users to create modeling languages collaboratively with NRT synchronization of edits. An illustration of the concepts and roles in the SyncMeta framework is given in Fig. 1. On the metamodeling layer metamodelers use the Meta-Model Editor for collaborative authoring of a metamodel, represented by a VLS. This builds the basis

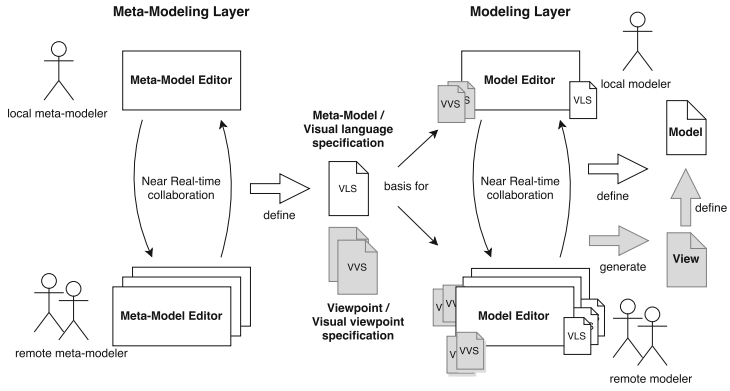


**Fig. 1.** Concepts and roles in the SyncMeta (meta-) modeling process [5], enhanced with views extension

for a model editor for the specified modeling language. A VLS is defined visually using a graph-based visual modeling language (VML). An arbitrary number of model editors can be generated based on the defined VLS. The NRT collaboration takes place at both metamodeling and modeling layers. The implementation details, architecture and interface offered by SyncMeta are detailed in the previous work [5]. The gray elements in Fig. 1 depict the view extension integrated into the SyncMeta framework. These also reflect the contributions of this work. On the metamodeling layer metamodelers may collaboratively define viewpoints in SyncMeta's metamodel editor. For the definition of a viewpoint the underlying VML was extended with additional view types. The view types define references to classes of the metamodel and offer to define conditions on the attributes of the referenced class (*RQ1*). Additionally the appearance and rules for each view type can be redefined in a view. To facilitate the metamodeling process a Closed-View Generation (CVG) algorithm based on [20] was implemented to automatically add classes and relationships to the viewpoint when a reference to an object or relationship is defined in the metamodel (*RQ1*).

Similar to the VLS generated for a metamodel, for each viewpoint a visual view specification (VVS) is generated which consists of a construction plan for the view in the modeling layer (*RQ2*). Here, an existing model can be used in combination with a certain VVS to generate a view on the model. Modelers may collaboratively edit any view or the model itself in NRT, with all actions being propagated to collaborators and reflected in all views and in the model (*RQ2*).

A simple example is given by the model-based community application design. Domain-specific experts from a certain community, software architects and software developers can define a metamodel for the information systems which should be developed in the respective community. For that, they create the metamodel collaboratively on the Web in NRT. Then, more VVS are defined in the metamodel, e.g. a view for frontend elements as modeling objects for community end users, a backend view for developers and a communication view between frontend and backend. Based on the defined VVS, a model editor is generated together with corresponding views. Community end users can collaboratively create the frontends they require on the frontend view, together with developers. Developers can give immediate feedback on the functionality required by end users. Developers can also edit in NRT the application backend and the communication between the backend and frontend, while architects can see in NRT the entire model and check the integrity of the modeled system. The view extension framework – following the same implementation policies of SyncMeta – is Web-based and fully open source (available in GitHub[1]).

## 3 Views and Viewpoints

The terms *view* and *viewpoint* are used interchangeably in many different reports and are often just introduced as examples. We therefore offer formal definitions for these terms and explain the relations between the different concepts used in

[1] https://github.com/rwth-acis/syncmeta

the visual modeling approach (*RQ1*, *RQ2*). The definitions are used in Sect. 4 for explaining the implementation of the viewpoint modeling and the view generation.

As in [7] a *viewpoint* is defined as a language which represents a metamodel. A viewpoint can restrict the original metamodel and it addresses a set of concerns of one or more stakeholders.

A *view* is the presentation of a model by applying a specific viewpoint. Thus, a view is a concrete instance of a viewpoint. A viewpoint is defined by a collection of view types.

A *view type* is a meta-class whose instances a view can display [8]. Thus, a view type is an object or a relationship class which comprises a set of rules. These rules can be "selectional" or "projectional" predicates that determine the representation of a object within the view.

In the following we introduce the formal definitions for the terms introduced above. First we define the sets of classes, properties and types and then define the formal concept of a metamodel.

Let $P$ be an infinite set of properties. Each $p \in P$ can be an arbitrary complex function or a simple value from an enumeration type. We only require that each $p$ has a label, a type and a unique identifier.

Let $T$ be the set of all types defined in the VML on the metamodeling layer of SyncMeta, e.g., $T = \{Object, Relationship, NodeShape, Generalization, Association, \dots\}$. An overview of all types in the VML is depicted in Fig. 3.

Let $C$ be an infinite set of classes. Any class $c \in C$ has a unique name, a type description, and a set of properties.

We define $\mathbf{label}(c) = l$ for $c \in C$. Analogously, we define $\mathbf{label}(C) = \{label(c) \mid c \in C\}$ as the set of all unique identifiers of all classes in $C$. Thus, we define the signature of a class $c$ as a triplet with $c = (l, t, A)$, where $l$ is the unique name of the class, $t \in T$ is the name of a type associated with the class, and $A \subset P$ a finite set of properties. We define $\mathbf{type}(c) = t$ as the type of class $c$. Analogously, we define $\mathbf{type}(C) = \{type(c) \mid c \in C\}$. Similar definitions can be found in [20].

**Definition 1.** *A **metamodel** is a directed graph $G = (V, E)$ with $V \subset C$ a finite set of nodes and $\forall c \in V : \mathbf{type}(c) \in T$. $E$ is a finite set of edges with $E = \{(l, t, c_i, c_j, A) \mid c_i, c_j \in V, c_i \neq c_j, t \in T, l$ an identifier, $A \subset P\}$.*

We assume that a metamodel may consist of an arbitrary number of classes and each class may consist of an arbitrary number of properties. We only require that the type of each class belongs to the VML. Analogously we can define a viewpoint. A viewpoint is a metamodel on its own. We just require that a viewpoint consists of at least one view type. Thus, we formally define a view type before we give a formal definition of a viewpoint. We define a function $\varphi$ that transforms an object class or a relationship class into a ViewObject or ViewRelationship class, respectively. On other classes the function $\varphi$ is the identity function.

**Definition 2.** *Let* $VT_C = \{\varphi(c) \mid c \in C\}$ *and* $\varphi(l,t,A) = (l',t',A',l)$ *with* $l'$ *the new unique label,* $t'$ *is ViewObject or ViewRelationship if* $t$ *is Object or Relationship, else* $t = t'$*. Obviously,* $A' \subseteq A \subset P$*.*

A view type class of a viewpoint consists of a reference to a class in the metamodel. The reference is the unique name $l$. Thus, a viewpoint is not independent of the metamodel.

**Definition 3.** *A* ***viewpoint*** *with respect to* $VT_C$ *is a metamodel with* $G' = (V', E')$*, and* $\exists c \in V' : c \in VT_C \wedge \textbf{\textit{type}}(c) = ViewObject$*.*

Based on these formal definitions of the concepts at the metamodeling layer we can define the concept of viewpoint applied to a model of the modeling layer of SyncMeta (*RQ1*). For the generation of the model editor instance a VLS of the metamodel is generated. For simplicity we think of the VLS as the metamodel described in Definition 1. First we formally define the relation between the metamodel defined in the metamodeling layer and the model.

**Definition 4.** *Based on graph* $G = (V, E)$ *of a metamodel, a* ***model*** *is a directed graph* $M = (V', E')$ *with* $\forall v \in V' : type(v) \in label(V)$ *and* $\forall e \in E' : type(e) \in label(E)$*.*

We require each node and each edge of the *model* to be an instance of a node type or edge type defined in the metamodel. To generate views we first need to define a function which applies a view type to an entity within the model:

**Definition 5.** *Let* $VP = (V, E)$ *be a VVS of a viewpoint. Let* $\phi_v(n)\colon (l,t,A) \mapsto (l, type(v), A')$*,* $v \in VT_V$ *is a view type class of* $VP$*.* $A' \subset P$ *is the new set of properties defined by view type* $v$*. Analogously, the function for edges is defined as* $\phi_v(e)\colon (l,t,c_1,c_2,A) \mapsto (l, type(v), c_1, c_2, A'))$*, where* $A'$ *is generated from the attributes defined for* $v$*.*

With this helper function we can define a view as follows:

**Definition 6.** *Let* $VP = (V_{VP}, E_{VP})$ *be a VVS of a viewpoint and* $M = (V_M, E_M)$ *a model. A* ***view*** $V = (V_V, E_V)$ *is a subgraph of* $M$ *with* $V_V = \{\varphi_v(c) \mid v \in V_{VP} \wedge c \in V_M\} \subseteq V_M$ *and* $E_V = \{\varphi_v(e) \mid v \in V_{VP} \wedge\ e \in E_M\} \subseteq E_M$*.*

The resulting view is a subgraph of the model it is applied on. Each node/edge whose type is referenced to a view type in the VVS is part of the view (RQ2). For the view generation we need a VVS and a existing model as input.

## 4 Architecture and Implementation

**Widgets.** Figure 2 depicts an overview of the widgets offered by SyncMeta with the view extension. The canvas widget visualizes the current state of the model and provides mechanisms to manipulate the model–e.g. adding nodes and edges, drag & drop, and similar. Each edit that alters the model is propagated locally

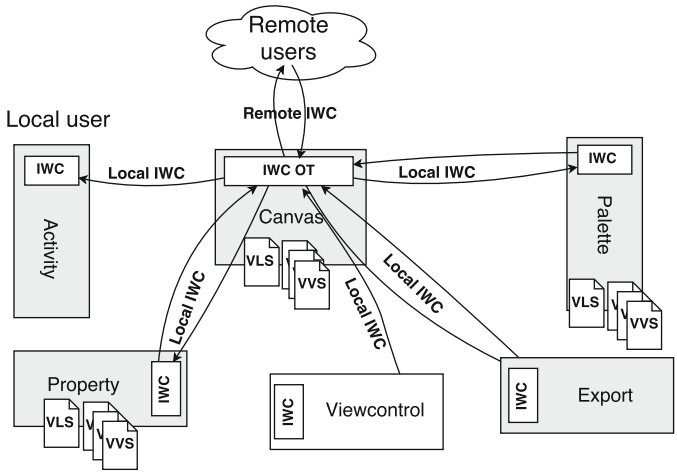


**Fig. 2.** Widget components of Syncmeta with the view extension

to other widgets and remotely to other collaborators. The property editor widget allows editing properties of node and edges selected in the canvas widget. Each property modification (e.g., changing the title of a node) is propagated back to the canvas widget. On the metamodeling layer the canvas widget saves and retrieves all nodes and edges of the metamodel and the viewpoints. The palette widget provides the nodes and edge types defined in the metamodel. Additionally the palette dynamically adjusts to the types defined in a particular VVS whenever a viewpoint is applied to a model. The activity widget tracks and displays the edits made by all collaborators. This is mainly for awareness purposes. SyncMeta consists of several additional widgets which serve special purposes, for example the export widget allows to export a model in JSON or PNG format. The view control widget allows to generate, export, and import a viewpoint metamodel or a VVS.

**Conflict Resolution.** SyncMeta enables non-locking collaboration— that is, each user can manipulate any part of the model at any time. The mechanisms to resolve editing conflicts are achieved using the OpenCoWeb JavaScript Operational Transformation (OT) Engine API [18], which is based on a decentralized peer-to-peer architecture. The details for the conflict resolution in the NRT modeling are given in [5] and are not repeated here due to space restrictions. The view extension uses also these mechanisms for modeling tasks and the view definitions. All operations are propagated to all other collaborators. At each receiving client the OT algorithms detect and resolve any occurring conflicts. The OT engine ensures a congruent model state after processing all operations at all client sides, following an optimistic approach (i.e. as opposed to approaches which use locking for all or parts of the model [3], changes are propagated in NRT andtherefore almost instantly visible at all sites).

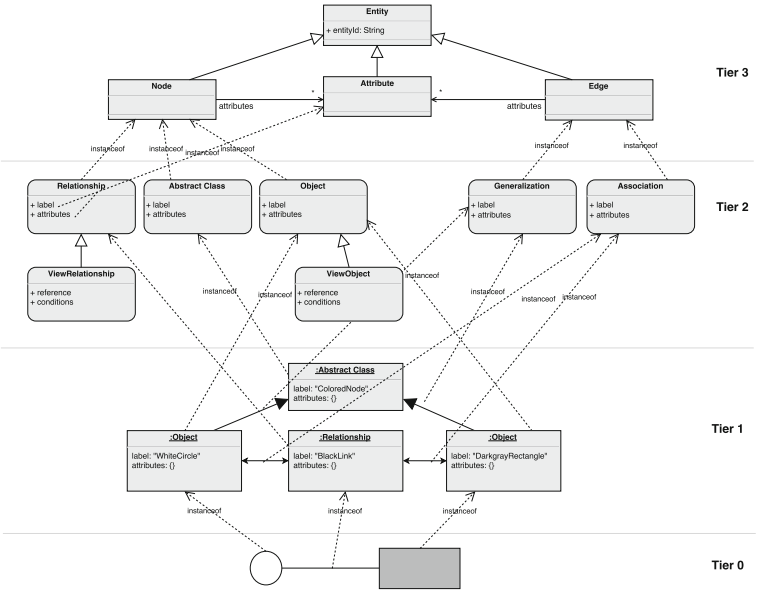


**Fig. 3.** Simplified extended metamodel hierarchy with view types

**Metamodeling and Viewpoint Modeling(*RQ1*).** In the previous section we have presented the formal definitions of viewpoints, views and view types and shown that we can apply the NRT collaborative modeling approach of SyncMeta also to the view extension. SyncMeta implements a four-tier metamodel hierarchy, which is depicted in Fig. 3. *Tier 3* defines the basic elements of a graph-based modeling language. *Tier 2* defines the node and edge types of the VML as well as the view types of the viewpoint models. As stated in Sect. 3, Definition 2, a viewpoint does not contain any *Object* or *Relationship* types. We replace them by using the *ViewObject* and *ViewRelationship* types, which are a specialization of *Object* and *Relationship*, respectively. These contain a reference to a node type or an edge type in the metamodel. It is also possible to define conditions on the attributes of the referenced class, i.e. in contrast to a simple object class a view-object offers functionalities to customize the attributes of a view. Metamodelers are able to hide and rename attributes. The *Conjunction* attribute determines the logical connector of the conditions. This can be either the logical AND or OR. Thus, we can build a formula with the predicates $\varphi_1, .., \varphi_n$, either with a conjunction over all predicates $\varphi_1 \wedge ... \wedge \varphi_n$ or with a disjunction over all predicates $\varphi_1 \vee ... \vee \varphi_n$. Conditions on attributes allow metamodelers to make simple queries on the attributes of an object class and filter the entities of this class in the view canvas of the model editor.

With auxiliary classes it is possible to define custom node and edge shapes for each view type. *Tier 1* defines the actual metamodel or viewpoint. Metamodelers are allowed to develop an arbitrary number of viewpoints in the same NRT

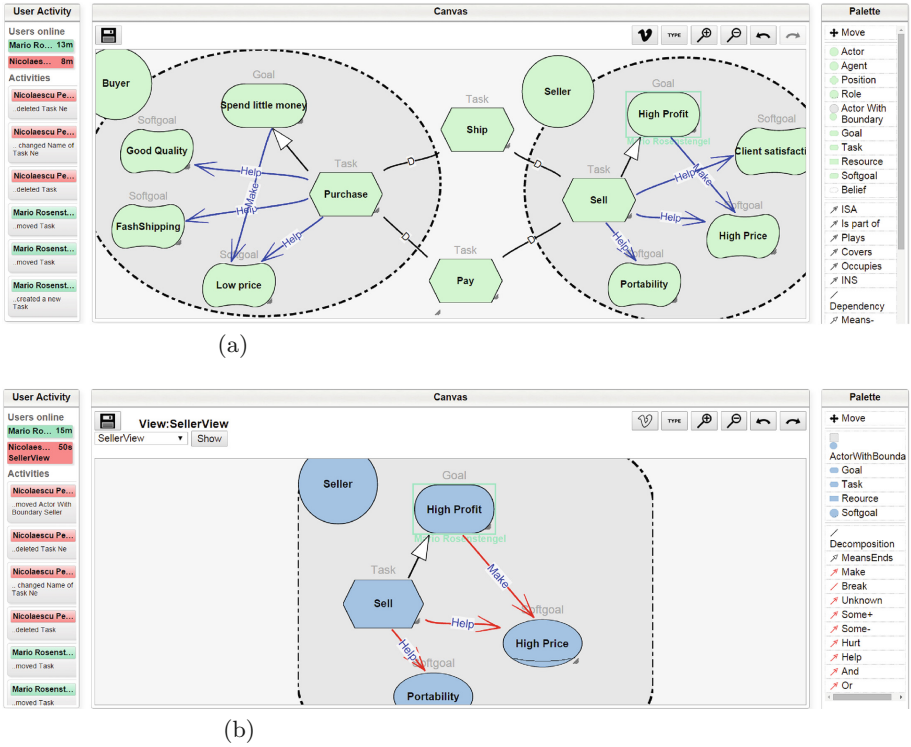


**Fig. 4.** (a) An $i^*$ model of buyer-seller relationships. (b) Applied *SellerView* to the model above

collaborative fashion they are used to define metamodels. The metamodel is the input for the model editor instantiation and each viewpoint is generated to a VVS. *Tier 0* is the actual model of the modeling layer. On Tier 0 a viewpoint is applied to the model. The resulting view supports NRT collaborative modeling as well. While concepts on Tiers 2 and 3 are implemented in the framework, models on Tier 0 and 1 are defined by modelers and metamodelers, respectively.

**View Generation(*RQ2*)**. On the modeling layer modelers may apply the viewpoints defined on the metamodeling layer. This is done for any existing model by selecting the desired view from a drop-down menu (see Fig. 4(b)). As described in Sect. 3, Definitions 5 and 6 all nodes and edges of the model that are associated with a view type in the viewpoint are then a part of the view, while all other nodes and edges are hidden. In addition to filtering on the type level, the framework also allows filtering nodes and edges on instance level based on the values of their properties. The selected view applies custom styles like adjusting the color, shape, labels or connectors. The following steps are used:

– Filter nodes/edges regarding the ViewObjects/ViewRelationships of the VVS
– Filter nodes/edges by conditions defined on their attributes
– Apply custom styles for each node/edge

Figure 4(a) shows a small $i^*$ model [21] about buyer-seller relationships, which was also used in the evaluation (see Sect. 5). Figure 4(b) depicts a possible view on the model, which is called "SellerView". It contains only nodes and edges within the boundary of the "Seller" actor, along with their edges. For demonstration purposes we also defined a slightly different styling for node and edge types. The palette widgets adapts to the view by only displaying node and edge types defined in view.

**Limitations.** As described in [5], the visual-based (meta-) modeling approach of SyncMeta has some restrictions, such as model checking functionalities on the (meta-)modeling layer. By extension, the views do not allow the specification of cardinalities or multiplicities with regard to the relationships and view-relationships. Also, it is not possible to define conditions on inherited attributes of super classes. Currently, only the definition of conditions for the attributes of the referenced class is allowed. A simple solution for this problem is that we define the attribute directly in the referenced class, but this is suboptimal and fails to exploit the inheritance hierarchy. Finally, the view-based modeling approach requires an automatic diagram layout mechanism. In the current implementation, a big disadvantage is that elements of a view are placed at the same position as in the model. Solutions to these limitations are planned to be implemented in future versions, since they are not critical for a research prototype.

## 5 Evaluation

We performed an end user evaluation of the model editor. The main goal was to evaluate the usability and usefulness of the view-based modeling approach and monitor the NRT collaboration features (*RQ3*).

**Participants.** The end user evaluation comprised four sessions with four participants each with a total of 16 participants, who were recruited from researchers and students of our department. Their expertise in conceptual modeling, $i^*$ and SyncMeta was measured using seven-point Likert scale (from 1=novice to 7=expert). The results show that users had varying existing knowledge of modeling. As such, expertise with graphical editors is quite high, but has a high standard deviation ($M = 4.38; SD = 2.42$). The same holds for user's general expertise in conceptual modeling ($M = 4.5; SD = 2.46$). However, the level of expertise with $i^*$ is rather low ($M = 2.44; SD = 2.5$).

**Methodology.** In each session, the four participants were split into two groups (*Group Alpha* and *Group Beta*) with two people each. Both groups had to complete two tasks of comparable scope. Each task comprised a list of detailed instructions to extend a given $i^*$ model with additional nodes and edges. This could be performed without any $i^*$ expertise. The collaborators could decide for themselves how to complete the instructions by communicating with each other via chat or just start modeling and let SyncMeta resolve potential conflicts. The first task was solved by Group Alpha and consisted of a predefined view applied,

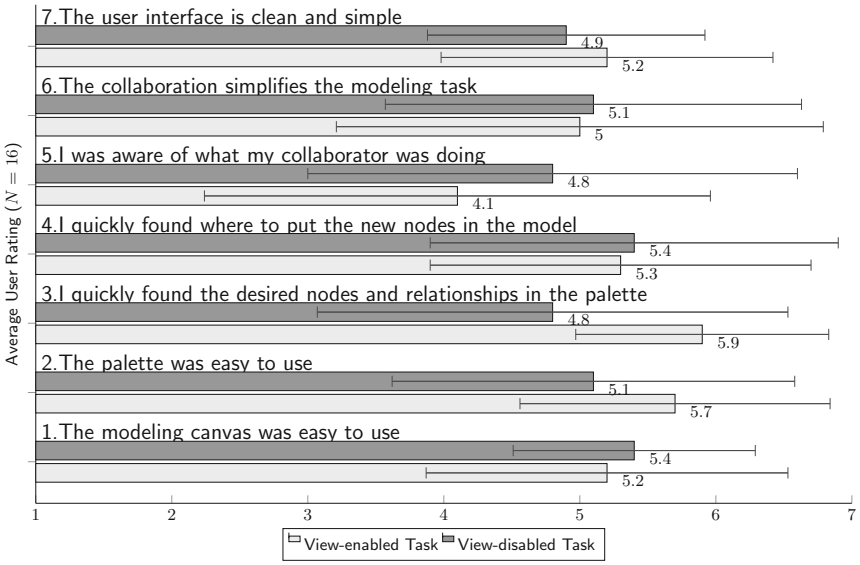


**Fig. 5.** Survey results of $i^*$ group sessions (quantitative items).

which customized the model editor regarding the requirements of the task (see Fig. 4(b)). Group Beta solved the same task without any view on the original model (see Fig. 4(a)). For the second task, they switched roles: Group Beta used a predefined view, while Group Alpha solved the task without a view.

After each task the session participants were asked to rate statements regarding their experience with and without views. The ratings were made using a seven-point Likert scale ranging from "strongly disagree" (1) to "strongly agree" (7). During the evaluation the working times for each task and group was recorded to determine whether the views had an impact on the time it took modelers to complete the tasks.

**Results.** The mean ratings for tasks solved with and without views are plotted as series "view-enabled task" and "view-disabled task", respectively, in Fig. 5. For most statements there is little difference between the ratings for view-enabled vs. view-disabled task. We ran paired-sample t-tests for view-enabled vs. view-disabled ratings to identify significant differences. Two statements exposed significant differences at $p < .05$, namely statement 3, revealing that views helped to find nodes and relationships quicker in the palette ($p = .01$), and statement 5, revealing that the views actually hampered the awareness of the collaborator's edits ($p = .04$)

Additionally the working times for each group and task were recorded. The average working time of Alpha groups for Task 1 without views ($M = 253\,\text{s}; SD = 39$) was on average 82 s or 52 % longer than the average working time for Task 2 with view enabled ($M = 171\,\text{s}, SD = 19$). The average working time of the Beta groups for Task 2 without views ($M = 191\,\text{s}; SD = 22$)

was on average about 24 s or 15 % longer in comparison to Task 1 with views enabled ($M = 167\,\text{s}; SD = 31$) . The lower improvement factor for the Beta groups compared to the Alpha groups can be explained by a learning curve. The Alpha groups used the views during the second task, where they were more familiar with how to work with the tool. Conversely, the Beta groups used the views during the first task when it was the first use with the tool for most of them. This actually shows that modeling with views speeds up the modeling process even for users who are unfamiliar with the tool (*RQ3*).

The findings are that views can improve user experience and speed up the modeling process; they can also be used to customize the model editor in order to ease adoption and to improve stakeholder involvement during the collaborative modeling process. Participants also provided some textual comments about the view-based modeling approach. They stated that they liked switching between views and that the reduced palette gives a better orientation, which may explain the faster modeling times with views enabled. The NRT collaboration features were already evaluated in SyncMeta [5], but challenges were also encountered. In the evaluation, NRT collaboration and edits awareness were only available between views and the entire model editor. However, the evaluation results have shown that users require also collaboration directly between individual views and this feature was implemented as consequence (*RQ3*).

## 6 Related Work

Table 1 demonstrates that views and related concepts have been successfully used in many research fields, including object-oriented databases (OODB) [2, 20], enterprise architecture (EA) [10,22] and corresponding frameworks and in conceptual modeling (CM) [1,6,9,12,13].

OODBs fully support general concepts of object-oriented programming languages. One of the most popular view extensions is called MultiView [2], a simple and powerful tool for supporting multiple views in the Gemstone OODB [19]. Multiview introduced the CVG-algorithm to facilitate the definitions of viewpoints. A similar approach is provided by our view extension (cf. Sect. 2).

EA frameworks are used to look at complex information systems from different point of view—e.g. data, function, networks, organizational, structures, schedules and strategy. The ARIS Framework [10] provides various model editors to build complex enterprise architectures, e.g. location allocation diagram, network diagram, technical resource model. All entities of these model editors are integrated into one comprehensive metamodel. The *Zachman Framework* [22] is a two dimensional classification schema for descriptive representations of an organization. It is an abstract guideline which proposes perspectives on a particular system of an enterprise in different development stages.

Finally, a plethora of CM tools also provide view extensions. *MetaEdit+* [9] is a tool set to define modeling languages and generate model editors. Unlike SyncMeta only a locking collaboration approach is used. *AToM*$^3$ [12] and *ADOxxx* [6] are domain-independent metamodeling frameworks with focus on

**Table 1.** Comparison of related tools and frameworks.

| Tool/Framework | Type | Domain-independent | Graphical view editing | Conditional filters | Conditional styles | Collab. viewpoint definition | Collab. view manipulation | NRT editing |
|---|---|---|---|---|---|---|---|---|
| Abiteboul &Bonner [2] | OODB | ● | | ● | | | | |
| Multiview [20] | OODB | ● | ● | | | | | |
| ARIS Framework [10] | EA | | ● | | | | | |
| Zachman Framework [22] | EA | ● | ● | - | - | | | |
| MetaEdit+ [9] | CM | ● | ● | | | ● | ● | |
| AToM$^3$ [12] | CM | ● | | ● | ● | | | |
| ADOxxx [6] | CM | ● | | ● | | | | |
| Sirius [1] | CM | ● | ● | ● | ● | | | |
| CO2DE [13] | CM | | ● | | | ● | ● | |
| SyncMeta Views | CM | ● | ● | ● | ● | ● | ● | ● |

simulation of models. *AToM*$^3$ allows to transform a model expressed in a certain formalism to an equivalent model in another formalism. ADOxxx [6] provides a query language called AQL for the generation of views on these models. In contrast to SyncMeta Views these frameworks do not provide any NRT collaboration features. *Sirius* [1] uses the Eclipse Modeling Framework (EMF) as basic infrastructure. It offers fully customizable viewpoints on complex models. Modelers can define conditional styles and filters for entities based on their attributes. It is possible to generate a subset of the available palette and define optional layers to show additional content. Sirius lacks NRT collaboration features, but it offers many customization options which makes the framework very powerful. *CO2DE* is a desktop collaborative modeling application. Similar to SyncMeta it provides awareness features to help users recognize edits of model elements and a chat room. CO2DE doesn't support metamodeling. However, it does not automatically solve editing conflicts. It uses a locking approach for enabling collaboration, which is therefore not in NRT. The philosophy is that users have to discuss about conflicts and deal with them on their own.

As this comparison shows, the views framework we implemented exhibits the key features for view definition, editing and use found in literature. As a highly distinguishing feature, SyncMeta Views enables non-locking NRT collaboration during view definition and use, which is not supported in any of the existing tools.

## 7 Conclusion and Future Work

In this work, we explored how metamodel-based view generation can be effectively combined with NRT collaboration in modeling for teams with different competences and roles. For this purpose, we presented a view extension for the SyncMeta metamodeling framework which allows the generation of views for focusing on particular aspects of a complete model. The views are editable and all edits are reflected in all views and in the model. Thus, we offer a unique approach of NRT collaboration for free conceptual model editing on the Web using optimistic concurrency control mechanisms, combined with known techniques for views definition and generation from information systems domain. The view-based extension was evaluated in group sessions using an instance of the $i^*$ language generated and initialized with a simple model and views. The evaluation results show that NRT collaboration for view-based authoring is possible, that by using views the modeling speed is slightly improved and that the views are useful for reducing complexity, especially when dealing with big models.

The view-based modeling proposed also opens many relevant new research directions. We plan to enhance the expressiveness of the conditions on a view type to allow more complex queries and model perspectives. Moreover, to improve the NRT collaboration features of the framework we have replaced the OpenCoWeb implementation and are currently evaluating SyncMeta Views with Yjs [14], a real-time P2P shared editing framework for arbitrary data types, as it overcomes scalability drawbacks and is much easier to use by developers. Furthermore, in order to improve the feedback during collaborative modeling and to support end-users working with the views extension, we are currently developing an intelligent assistant system for collaborative modeling scenarios to guide collaborators during the modeling process using different strategies like remote support or conflict avoidance. Together with an automatic distributed approach to deal with co-evolution of metamodels and models, these improvements will gear the framework towards use in real-world information systems engineering projects.

**Acknowledgments.** This research was co-funded by the European Commission through the FP7 Integrated Project "Learning Layers" (grant no. 318209).

## References

1. Sirius - The easiest way to get your own modeling tool: Graphical Editors for your DSL (2014). https://www.eclipse.org/sirius/features.html
2. Abiteboul, S., Bonner, A.: Objects and Views. In: York, A.N. (ed.) ACM International Conference On Management Of Data (SIGMOD), pp. 238–247. ACM, New York (1991)
3. Chechik, M., Dalpiaz, F., Debreceni, C., Horkoff, J., Rath, I., Salay, R., Varro, D.: Property-based methods for collaborative model development. In: Joint Proceedings of the 3rd International Workshop on the Globalization Of Modeling Languages and the 9th International Workshop on Multi-Paradigm Modeling, pp. 1–7 (2015)

4. De Lange, P., Nicolaescu, P., Derntl, M., Jarke, M., Klamma, R.: Community application editor: collaborative near real-time modeling and composition of microservice-based web applications. In: Modellierung (2016)
5. Derntl, M., Nicolaescu, P., Erdtmann, S., Klamma, R., Jarke, M.: Near real-time collaborative conceptual modeling on the web. In: Johannesson, P., et al. (eds.) ER 2015. LNCS, vol. 9381, pp. 344–357. Springer, Heidelberg (2015). doi:10.1007/978-3-319-25264-3_25
6. Fill, H.G., Karagiannis, D.: On the conceptualisation of modelling methods using the ADOxx meta modelling platform. Enterp. Model. Inf. Syst. Architect. **8**(1), 4–25 (2013)
7. Fischer, K., Panlenko, D., Krumeich, J., Born, M., Desfray, P.: Viewpoint-Based Modeling - Towards Dening the Viewpoint Concept and Implications for Supporting Modeling Tools (2012)
8. Goldschmidt, T., Becker, S., Burger, E.: Towards a tool-oriented taxonomy of view-based modelling. In: Sinz, E.J., Schürr, A. (eds.) Modellierung (2012)
9. Kelly, S., Lyytinen, K., Rossi, M., Tolvanen, J.P.: MetaEdit+ at the age of 20. In: Bubenko, J., Krogstie, J., Pastor, O., Pernici, B., Rolland, C., Sølvberg, A. (eds.) Seminal Contributions to Information Systems Engineering, pp. 131–137. Springer, Heidelberg (2013)
10. Kozina, M.: Evaluation of ARIS and zachman frameworks as enterprise architectures. J. Inf. Organ. Sci. **30**(1), 115–136 (2006)
11. Kurpjuweit, S., Winter, R.: Viewpoint-based meta model engineering. In: Reichert, M. (ed.) Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures, GI-Edition/Proceedings, vol. 119, pp. 145–158. Ges. für Informatik (2007)
12. de Lara, J., Vangheluwe, H.: $AToM^3$ : a tool for multi-formalism and meta-modelling. In: Kutsche, R.-D., Weber, H. (eds.) FASE 2002. LNCS, vol. 2306, p. 174. Springer, Heidelberg (2002)
13. Meire, A.P., Borges, M., de Araújo, R.M.: Supporting multipleviewpoints in collaborative graphical editing. Multimedia Tools and Appl. **32**(2), 185–208 (2007)
14. Nicolaescu, P., Jahns, K., Derntl, M., Klamma, R.: Yjs: a framework for near real-time P2P shared editing on arbitrary data types. In: Cimiano, P., Frasincar, F., Houben, G.-J., Schwabe, D. (eds.) ICWE 2015. LNCS, vol. 9114, pp. 675–678. Springer, Heidelberg (2015)
15. Nissen, H.W., Jarke, M.: Repository support for multi-perspective requirements engineering. Inf. Syst. **24**(2), 131–158 (1999)
16. Nissen, H.W., Jeusfeld, M.A., Jarke, M., Zemanek, G., Huber, H.: Managing multiple requirements perspectives with meta models. IEEE Softw. **13**(2), 37–48 (1996)
17. Nuseibeh, B., Kramer, J., Finkelstein, A.: ViewPoints: meaningful relationships are difficult!. In: Proceedings of the 25th International Conference on Software Engineering, 2003, pp. 676–681 (2003)
18. OpenCoWeb: Open Cooperative Web Framework 1.0 Documentation
19. Rundensteiner, E.A., Kuno, H.A., Ra, Y.G., Crestana-Taube, V., Jones, M.C., Marron, P.J.: The MultiView project. ACM SIGMOD Rec. **25**(2), 555 (1996)
20. Rundensteiner, E.A.: MultiView: a methodology for supporting multiple views in object-oriented databases. In: Kaufmann, M. (ed.) Proceedings of the 18th VLDB Conference, pp. 187–198. Morgan Kaufmann (1992)
21. Yu, E.: From organization models to system requirements: a 'cooperating agents' approach. In: Cooperative Information Systems (1995)
22. Zachman, J.A.: The Zachman Framework: A Primer for Enterprise Engineering and Manufacturing. Zachman Framework Associates, Toronto (2003)