UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS Honours Project Final Paper 2024

Title: TRENDy - Designing a graphical modelling tool for the TREND Temporal Conceptual Data Modelling Language

Author: Richard Taylor

Project Abbreviation: TRENDy

Supervisor(s): Prof. M Keet

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	15
Theoretical Analysis	*0*	*25*	0
Experiment Design and Execution	*0*	*20*	5
System Development and Implementation	*0*	*20*	20
Results, Findings and Conclusions	*10*	*20*	10
Aim Formulation and Background Work	*10*	*15*	10
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	
Total marks		**80**	**80**

TRENDy: Designing a graphical modelling tool for the TREND Temporal Conceptual Data Modelling Language

Richard Taylor
University of Cape Town
Cape Town, Western Cape, South Africa
tylric007@myuct.ac.za

ABSTRACT

This paper presents the development of a graphical modeling tool specifically designed for the TREND Temporal Conceptual Data Modeling Language. While temporal data modeling is increasingly vital for applications like historical data tracking and forecasting, no dedicated tools currently exist to support the unique requirements of temporal conceptual models. As a result, users are forced to rely on generic modeling tools, which can limit productivity and increase the barrier to entry for beginners. Our tool addresses these challenges by providing an intuitive, web-based interface tailored to TREND, incorporating features such as dynamic extension and evolution, schema generation, and model validation. By offering specialized support for temporal elements and constraints, this tool aims to streamline the modeling process, enhance usability for novice and experienced modelers alike, and set a standard for future temporal data modeling tools. After evaluating this tool through task based evaluations, it is observed that this newly developed tool, called TRENDy, is consistently ranked as more usable than current solutions, and is also more efficient and results in less modeller mistakes. From this, along with implementing all required features from expert modellers, this paper concludes this tool is successful in the goals it sets out to achieve.

KEYWORDS

Temporal data, Conceptual data modelling, graphical tooling

1 INTRODUCTION

As data storage cost has decreased over time, the desire and ability to capture historical data has increased, and thus the necessity for temporal databases has risen. Temporal data is particularly vital for a wide range of applications, including business process modeling, trend analysis, video surveillance [?], clinical data analysis [3], forecasting, and adherence to legal and regulatory standards [23]. Since the early 1980s, there has been growing interest and research in temporal databases, and by the early 2000s, most academic and research database applications incorporated temporal data [12]. Temporal databases store data that is referenced by time [13]. Unlike atemporal databases, which assume that all stored data is currently valid, temporal databases maintain data that includes one or more time periods during which the data is valid [18].

Temporal Conceptual Data Modeling Languages (TCDML) are used to represent temporal data. When we create conceptual data models, we aim to develop simplified representations of our information domain to better understand the problems at hand. The standard conceptual modeling language for atemporal databases is the Entity-Relationship (ER) Diagram (not like the Unified Modeling Language, which focuses on implementation rather than conceptual modeling).

The structure of a temporal database can be modeled using a TCDML. Most existing TCDMLs build upon the ER Diagram, extending it with specific temporal features [1, 2, 11, 22]. These TCDMLs provide a high-level conceptual overview of the logical structure underlying temporal databases, presented in a way that is accessible either textually or graphically.

Temporal Information Representation in Entity-Relationship Diagrams (TREND) is a temporal extension of the Extended Entity-Relationship Diagram (EER). EER, an augmented version of the ER diagram, adds functionalities such as inheritance and generalization [15]. TREND is a TCDML that emphasizes usability and accessibility [4, 16], and achieved this through extensive user testing that took form in 11 experimentation's.

TREND incorporates temporal extensions using two key constraints: dynamic extension and dynamic evolution [4]. These constraints define how temporal entities in a model can change over time. With dynamic extension (EXT/ext), the extended entity remains an instance of the original entity. For instance, an *Employee* who gets promoted to *Manager* still remains an *Employee*. In contrast, dynamic evolution (CHG/chg) occurs when the extended entity is no longer the same instance as the original entity. For example, when an *Article Clerk* in a law firm becomes a *Lawyer*, they cease to be an *Article Clerk* [4]. Additionally, several key distinctions in notation are important for TREND:

- **Icons:** Temporal entities as described above are marked with a clock icon, while atemporal entities (legacy ER/EER entities) are not.
- **Mandatory or Optional**: Mandatory temporal constraints (those that must have occurred in the past or will occur in the future) are shown with solid lines, while optional constraints (those that might have occurred in the past or may happen in the future) are represented by dashed lines.
- **Past or future**: Lowercase letters (e.g., "chg," "ext") indicate past conditions, while uppercase letters (e.g., "CHG," "EXT") indicate future constraints.
- **Quantitative**: When a constraint has a quantitative value attached (e.g., an *Employee* must have been an *Assistant Manager* for two years before becoming a *Manager*), the value is shown in time units next to the constraint type.

For further clarification, this notation is summarized in Figure 4, in the appendix.

Despite the existence of numerous conceptual data modeling tools for general purposes, none are specifically tailored to the TREND conceptual modeling language. As a result, if a modeler wishes to design a temporal conceptual model, they are forced to use generic tools or rely on pen and paper. This increases the likelihood of errors, as the wide variety of shapes and relationships in general-purpose modeling tools can lead to inconsistent notations or the incorrect use of elements. Furthermore, there are no TREND-specific features for verbalization or validation to guide users. This raises the barrier to entry for beginners and slows down productivity for both novice and expert users. In the related work section, we will explore other graphical tools in this domain.

2 RELATED WORK

Before designing and implementing the project, three similar open-source graphical modelling tools for other conceptual data modelling were analysed: ERD Tool, Crowd and Draw.io. The successful approaches that pertain to our application domain will be a base of inspiration for our graphical TREND tool. Screenshots of these tools' user interfaces can be found in the appendix.

2.1 ERD Tool

The Entity-Relationship Diagram Tool (ERD Tool) is designed for graphical modeling of ER Diagrams [20]. It emphasizes a simplified and streamlined user interface while covering the essential requirements of ER Diagrams, such as cardinality, participation, and recursive relationships [20]. Although these features do not meet the specific needs of our application, examining the underlying framework used by this tool can provide valuable insights for applying similar structures to our domain.

The Graphical Editing Framework (GEF) is particularly useful for tasks like configuring models, saving them, and creating a palette for users. This palette includes the necessary tools for model building, such as attributes, entities, and their relationships [20].

GEF implemented the Model-View-Controller design pattern, where the canvas the the user sees is connected to the underlying model data through the use of a controller, so if the user changes a model or canvas, the controller will update the other accordingly. In this design, there is a separate controller for each type of object on-screen (entity, attribute and relationship). GEF also supports notifications with the use of the observer design pattern. These design decisions will be noted when designing a similar tool. This tool does ERD validation (its own specific grammar), but this validation is not extensive. If a user makes an obvious mistake, such as connecting two attributes, they will be notified, however this error checking does not apply to more sophisticated ERD rules. Additionally, this product misses our on much of the usability testing and features users would expect, such as snap-to geometry and view-port controls [20].

GEF employs the Model-View-Controller (MVC) design pattern, where the user's canvas is linked to the underlying model data through a controller. If the user modifies the canvas (model), the controller ensures the other is updated accordingly. In this pattern, each on-screen object (entity, attribute, or relationship) has its own controller. GEF also incorporates notifications using the observer design pattern. These design principles will be considered when developing a similar tool. While the tool does offer ERD validation based on its own specific grammar, this validation is limited. It will notify users of basic errors, such as connecting two attributes, but it doesn't cover more complex ERD rules. Additionally, the tool lacks usability features like snap-to geometry and view-port controls, which are typically expected by users [20].

2.2 Crowd

Crowd is a research tool that facilitates temporal ER modeling, enhanced by automated reasoning. It operates using a client-server architecture, where each client implements the Model-View-Controller (MVC) design pattern and interacts with a server for reasoning support [10]. The tool's reasoning capabilities include checking for class and relationship consistency as well as discovering implied class and cardinality constraints [5]. While this is more advanced than the ERD Tool, it still lacks the specific temporal constraints that we aim to implement. However, we could adapt a similar approach to apply our own temporal constraints.

Earlier versions of Crowd use OWLlink to interface with OWL (Web Ontology Language) reasoners, enabling graphical reasoning support through a Knowledge Base (KB), which is a collection of axioms and facts that must be enforced [19]. OWL allows for precise definitions of vocabulary terms and their relationships, a process known as creating an ontology [21]. However, no version of Crowd currently supports constraint checking for temporal extensions to ER diagrams. Only Crowd 1.0 supports building temporal ER extensions, while the latest version, Crowd 2.0, supports EER, UML (Unified Modeling Language), and ORM 2, with reasoning capabilities provided by the RACER and Konlcude reasoners [10]. To develop our TREND tool, we could extend the rules in Crowd 2.0 to include support for temporal graphical objects. The latest version of Crowd features an updated user interface with essential improvements to make the tool more comparable to advanced modeling systems. These features include draggable objects with snap-to-geometry functionality, allowing users to create relationships and specify cardinalities between them. Additionally, it offers an automatic layout feature, which arranges objects in various layouts such as grid, circular, and concentric formats, along with zoom control. However, some limitations remain: the reasoner can be unreliable, and newer versions lack temporal features, meaning the palette does not include any temporal objects. Crowd also lacks support for keyboard shortcuts.

2.3 Draw.io

Draw.io (formerly diagrams.net) is a web-based tool (also available as a desktop version) used for creating a variety of diagrams, including flowcharts, ER and UML diagrams, and network diagrams. Its open-source nature makes it a candidate for extending into the specialized tool we require. Among existing modeling tools, Draw.io boasts one of the most polished, feature-rich, and extensive user interfaces. It offers robust customization options for graph styles (text, color, and layout), premade templates for modelers, and a comprehensive palette of objects. However, when users select "Entity-Relationship Diagrams," they are presented with a template that more closely resembles a UML class diagram than a proper ER diagram. The tool also fully utilises shortcuts, allows for mouse drag-to-select (to group and move objects simultaneously), and supports inserting custom-made objects and images. Additional features include an integrated spellchecker, broad language support, automatic backups, find-and-replace functionality, plugin support, zoom, and layering options to occlude objects. This extensive feature set makes Draw.io an ideal candidate for extending into our desired tool, as it is state-of-the-art in graphical modeling. One significant drawback of Draw.io is its lack of support for model validation and the broad range of notations it offers, which can lead to user errors. For instance, as shown in Figure 7 in the appendix, loading an ER diagram results in UML classes being added, demonstrating the inconsistency between different modeling languages. Additionally, the tool lacks support for temporal constraints.

OntoPanel [7] provides valuable insight into how reasoning extensions for Draw.io could be implemented. It is a plugin designed to assist domain experts in creating and mapping visual ontologies for materials testing, ensuring compliance with FAIR principles (Findable, Accessible, Interoperable, and Reusable). OntoPanel includes tools for importing ontologies, converting diagrams to OWL, verifying them against OWL rules, and mapping data for interoperability [7].

The front-end, built in JavaScript, interacts with Draw.io's mxGraph library, while the back-end, developed using the Django REST framework, converts models into a format that OWL reasoners can check for constraint violations, with errors sent back to the user [7]. This approach offers a blueprint for integrating TREND temporal reasoning into our tool using Draw.io as a base, with potential future features like controlled natural language conversion [14] or hand-drawn model translation [9].

From analysing the related tools, we can conclude that there is a need for a novel graphical tool to assist in the creation of temporal models. Despite the existence of various tools for general conceptual modeling, none fully meet the specialized needs of temporal modeling with intuitive graphical interfaces and comprehensive reasoning support. Specifically, no tool exists for modelling in TREND, thus beginners and experts alike are forced to either use pen and paper or draw their models in a generic modelling tool with no validation or verbalization support.

3 DESIGN

The design for this tool was informed by similar tools which would be considered state-of-the-art in the domain of conceptual data modelling. The design was also informed by our expert modellers (namely the project supervisors and my project partner) and from feedback from beginner modellers. This has been distilled as an (ordered by importance) list of features the tool would need (Table 1). Having input from both expert and beginner modellers allows the design of the tool to be both accessible to new conceptual modellers, but enough depth of features to be useful to more experienced modellers. Everything up until feature 4 will be considered absolutely essential to the project. Features 5 to 9 are considered important, but not essential, and features 10 to 13 are quality-of-life features that will still improve the overall experience of using this tool, but are considered as less important. Non-functional requirements will form part of this less essential, yet still important, as a failure in a non-functional feature such a performance or reliability would decrease the overall quality of the tool, but not render the tool obsolete. Due to hard limit on time and resources for this project, as well as the dynamics feature, it will make the most sense to choose an agile software methodology. This also allows for better flexibility and adaptability of the project, and give more opportunities for user/stakeholder engagement.

3.1 Architecture

The project is designed using model-view-controller. This pattern is robust [17]. The user interface will be the implementation of the view, and the serialization will be implemented as the model (this will be discussed in detail in the implementation section). The model must also be separate from the view and controller to allow for feature 3 (Ability to load and save models) to be implemented. The specific implementation of such a program will be discussed in the next section, but as it pertains to the design, the model serialization is designed as a JSON object. It contains separate lists of nodes and edges, which here represent the entities or objects in our model and the relationships between them respectively. This design choice was motivated by the structure of Stephan's schema generation tool, a similar serialization was picked to ensure easy and seamless integration of our tools. Additionally, the design choice of using a JSON object integrated well with the react library used for the front-end, and allowed for an easy way to check model structure to check for model validation (again, this will be discussed further in the implementation section).

The following tables describe the node and edge data types: In

Table 2: Attributes of the Node Object

Name	**Type**	**Explanation**
ID	*string*	A unique identifier.
Type	*string*	Always a generic shape type, since the data object stores the specific type of shape.
Position	*int, int*	x and y position on the canvas.
Data	*string, string, boolean*	An object containing styling data about the node, including specific type (e.g., Frozen Attribute, Temporal Entity), label, and if the node is an identifier.
Style	*int, int*	Width and height of the node.
Selected	*boolean*	Indicates if the current node is selected.
Dragging	*boolean*	Indicates if the current node is currently being re-positioned.

the appendix is a full example of the serialization described. The class structure is shown in Figure 2.

3.2 Description of Class Diagram

The class diagram below further describes the design of this tool.

- The **Reactflow App** class is the main class that runs, and it helps connect all aspects of the program together. This includes managing things such as file saving, keyboard shortcuts.
- The **Initial** Elements class contains the first nodes and edges when the tool is opened.
- Undo Redo handles state management (i.e undo and redo features), and is used by the main **Reactflow App**.
- **Verbalization** is the class that handles model to text Controlled natural language verbalization, and is used by the main **Reactflow App**.

Table 1: Feature Requirements table

Rank	Feature	Explanation
1	Full basic graphical modelling functionality	Ability to graphically basic ER diagrams using my user interface, including the ability to drag and drop objects from the palette to a canvas where these objects can be moved, and the ability to rename objects on screen and link them together.
2	Full TREND support	TREND graphical features including full extended entity relationship support in addition to temporal elements (attributes, entities, relationships) and dynamic extension & evolution. Implemented using a palette of SVGs rendered on-screen.
3	Ability to save and load models	The model information must be independent from the view and controller such that one can save models and load them later so they can save progress and work on them later.
4	Schema Generation	The user will have the ability to generate a database schema based off their model, this part of the project will be developed by my project partner Stephan.
5	State Management	The ability to undo and redo actions made in the program.
7	Graph Validation	This feature will be useful in helping new modelers with TREND. If the graph is edited, it will validate if the current state of the graph is in accordance with TREND; if not, a relevant error will be thrown to the user.
8	Edge Verbalization	To aid in understanding with TREND temporal constraints, the natural language translation of the edge should be displayed when the edge is selected.
9	Export to different formats	The graphs should be able to be exported as more useful formats for the user, such as PNG or JPEG
10	Shortcut support	Support of common and expected shortcuts, such as: CTRL+A, CTRL+C, CTRL+V, CTRL+X, as well as some new shortcuts that are specific to our program.
11	UI and viewport features	The canvas that users interact with to create their models should support snap-to-grid and be infinitely expandable in every direction, users should be able to resize and rotate objects on the canvas, and the final product should include an overview map that shows a zoomed-out high-level view of the entire model to assist the user in navigating their model.
12	Notation Conversion	Users should have the ability to convert between the specific EER notation they want to use.
13	Non-functional Requirements	The tool must also satisfy the non-functional requirements of usability and performance - the tool should be able to be used by modellers of all experience types and be able to handle models of large complexity without slowing down or crashing.

Table 3: Attributes of the Edge Object

Name	**Type**	**Explanation**
Type	*string*	Inherited edge type.
Style	*string, int*	Style information for the edge, including color and width.
Source	*string*	Source node ID.
Source Handle	*string*	Specifies which handle on the source node the edge is connected to.
Target	*string*	Target node ID.
Target Handle	*string*	Specifies which handle on the target node the edge is connected to.
ID	*string*	Unique identifier for the edge.
Data	*[string]*	List of errors, if any, associated with the edge.
Selected	*boolean*	Indicates if the edge is currently selected.

- The **Leva Control Panel** class is a component the forms part of the **Reactflow App**, and is used to display and handle global variable settings for the user.
- The **Driver** class acts as an entry point into Stephan's schema generation tool.
- The **ShapeNode** class is a component that forms part of the **Reactflow App**, and is used to represent a generic **ShapeNode** object that is rendered on-screen when dragged and dropped from the sidebar. This **ShapeNode** is extended by all the specific Shapes we want to render to the screen. This includes **Weak Entity**, **Weak Relationship**, **Temporal Relationship**, **Temporal Entity**, **Temporal Attribute**, **Derived Attribute**, **Atemporal Relationship**, **Atemporal Entity**, **Inheritance**, **Frozen Attribute**, **Atemporal Attribute**.
- The **Sidebar** class is a component that forms part of the **Reactflow App**, and is used to render a palette of ShapeNodes that can be dragged and dropped onto the screen.
- The **Edges** class is a component that forms part of the **Reactflow App**, and and is used to represent a generic **Edge** object that is rendered on-screen, which can be connected between two **ShapeNodes**. An edge is extended by the specific Edge Types: The **Atemporal Edge**, **Inheritance Edge** and the **Temporal Edge**.
- The **Edge Validator** is a class used by the **Edges** class to check temporal constraints over all the different edge connections.

The code structure was chosen for its robust and expandable nature. Allowing for easy long-term maintainability and scalability which

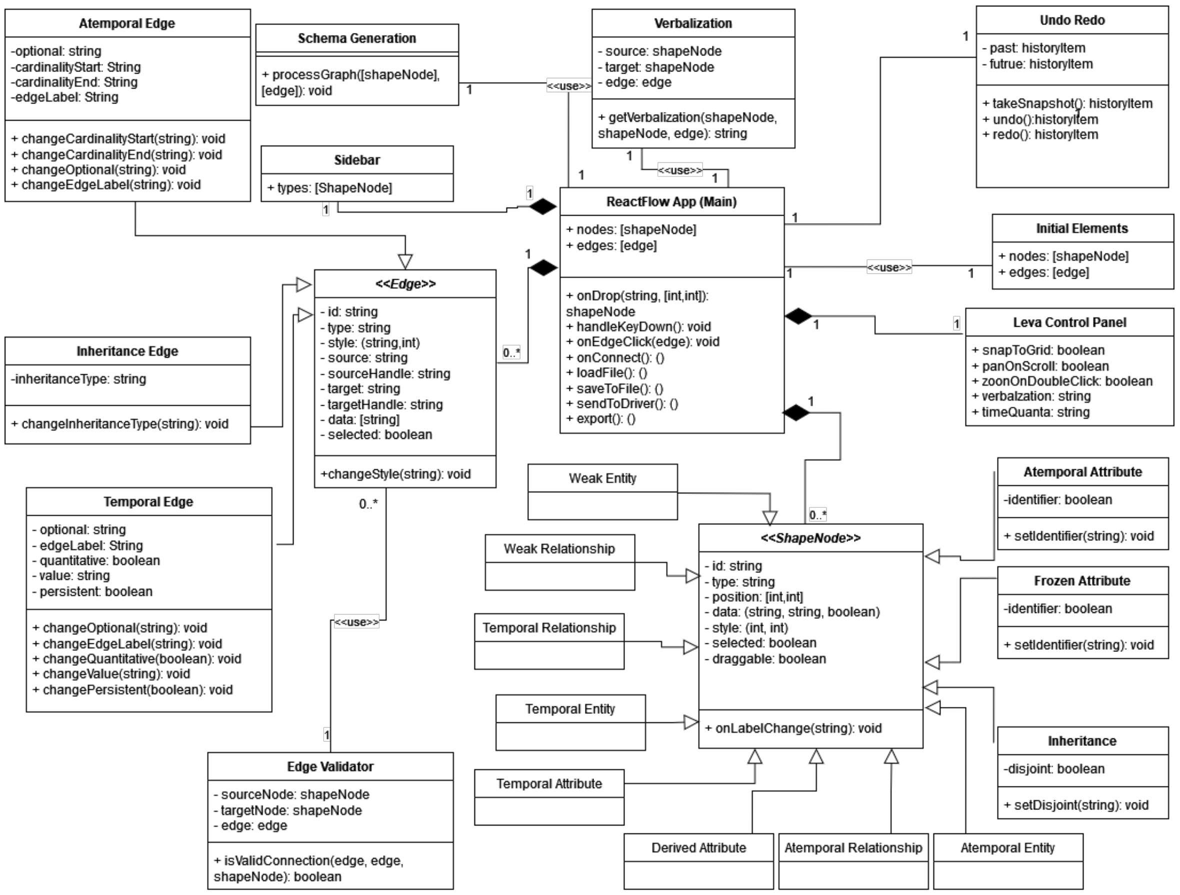


Figure 1: Class Diagram of the TRENDy tool's code architecture

effectively future-proofs the tool. For example, if new shape types are added to TREND, these can easily be added by creating a new extension to the generic **Shape Node** component, and it will be automatically added to the **Sidebar** component to be used by the user.

3.3 Requirements gathering

During the process of the building of this tool, meetings were the two supervisors Professor Maria Keet and Professor Sonia Berman, and the developer of the adjacent schema generation tool, Stephan Maree. From these meetings, feedback was obtained, specifically aimed at improving the experience for experienced modellers. This often came in the form of suggesting new features that makes the modelling process easier and more efficient for experienced modellers. This includes keyboard shortcut support, save and load features (to allow modellers to come back and complete their work at a later stage), and some of the view-port features. These meetings also acted as an early opportunity to obtain expert input from the supervisor and second reader, and catch out obvious bugs earlier in the stage of development. These meetings were originally scheduled online using Microsoft teams during the first 2 sprints, but the last 3 sprints were done in-person. Beginner user testers also informed features of the design towards the latter part of development, to help make the application more accessible through user testing feedback (this is further discussed in the evaluation section).

4 IMPLEMENTATION

4.1 Methods and Materials

For the implementation of the tools, it was decided to construct a web-app. This allows greater accessibility, since now the application is available cross-platform, without needing a previous native installation. Additionally, all scripting in the tool is done client-side, and this gives the tool greater accessibility as it can be used with a poor internet connection, since the tool is loaded in once and then can be used for an extended period without needing to connect to the server to run any scripts. The view and controller were handled by ReactFlow, which is a react library designed for node-based UI design. This node-based approach works perfectly for our use case, where we can abstract the concept of TREND entities, relationships and attributes as Reacflow Nodes, and abstract the TREND temporal constraints, and ER connections and ReactFlow edges. The library was chosen for its continued developer support and open-source nature. The code utilises the MIT open-source licence, which allows us to use their library for free provided we do not use this tool for commercial purposes. As the library is a react library, the tool is mainly coded in JavaScript, TypeScript and XML. As explained in the design section, the model was implemented using a JSON serialization, with separate list describing all the nodes and edges in the model. The motivation for this design choice was for easy integration with the react front-end and the schema generation tool, and also for a simple way to check for validation over the model.

The tool was designed using the agile methodology, where the development was divided up into five two-week sprints (since the total development phase of this project was 10 weeks), and each sprint was concluded with a supervisor meeting to check progress and provide feedback. The first sprint focused on the very basic and most vital features of implementing an infinite canvas, and adding the ability to drop objects into the canvas.

In the second sprint, the ability to resize and rename nodes was added, as well as the ability to connect nodes, and save and load models. The infrastructure of model validation was implemented, but no rules were added.

In the third sprint, the state management feature was implemented, and all the keyboard shortcuts were added. The ability to export models as images and other miscellaneous user interface features were added.

In the forth sprint, TRENDy was integrated with the schema generation tool, and the model validation rules were added. Additionally the verbalization feature was fully implemented.

The fifth sprint was used to acquire user feedback, and then implement the new features and improvements from users. This was also a period of bug-fixing.Due to a late Department of Student Affairs approval, full user acceptance testing and ’beginner’ client driven development was postponed to the last sprint, so most of the client driven development was provided by the ’experienced’ modellers. Despite this, ’beginner’ user feedback was still taken in after the user testing, and changes to the tool were made.

4.2 Implemented Features

This section contains a comprehensive list of final program features that were implemented to satisfy the design requirements.

4.2.1 Infinitely expanding canvas. As as space to build your models, an infinitely expanding canvas has been added, where nodes and edges can be dropped, moved, and edited. Basic zooming and panning functionalities have also been added to navigate the canvas

4.2.2 Dragging and dropping of TREND compatible objects. As part of implementing the conceptual data modelling functionality, a shape node palette has been added. This contains all the necessary entities, attributes, and relationships you need to model in TREND. An advantage of this palette over the palettes of alternative tools, is there is no way to added objects that are not part of TREND. This restriction prevents many user errors.

4.2.3 Resizing and Renaming of Nodes. Once a node has been added to the canvas, you can change the label of the node to give it your desired name. Additionally, if your label is too long for the node, you can resize the node to your liking too.

4.2.4 Ability to connect objects. Each object added to the screen has 4 drag handles (top, bottom,left and, right). When dragging from one of these drag handles to another, an edge is made to connect these objects together.

When selecting an edge, an edge option menu is displayed. This menu has all the necessary options on it to customise the edge to any edge the user desires in the TREND language. There are three main edge types: Atemporal (default), temporal, and subsumption. After selecting an edge type, the user then has access to other options to further customise their line (e.g. cardinality and persistence).

4.2.5 Schema Generation Tool. This feature needed to be able to convert the models we have developed in the user interface directly into a database schema. This feature was actually developed by my project partner, Stephan Maree, so for this tool, integration with the schema generation tool was necessary. A button called *Schema Generation* is added to the UI, and when pressed, it sends a JSON object with the nodes and edges serialization (described in the design section) to the schema generation tool, which then controls the conversion, and downloads the schema for the user. Since our tool used almost identical serializations, the JSON object can be used by the schema generation tool with little processing done on the receiver side.

4.2.6 Saving and loading of models. Due to the Model-view-controller design of this tool, the model information is independent from the rest of the code. This allows the program to add the ability to load and save models. When clicking the SAVE button, the tool creates a JSON object and saves it to the user’s computer. When clicking the BROWSE button, the user is given a prompt to select the file they would like to upload, and then the tool loads in the JSON object to the canvas. If the model that is uploaded to the program has an incorrect format (i.e. it was edited outside of the graphical tool) then the tool will do it’s best to display as many correctly formatted objects on-screen as possible, and displays a relevant error to the user.

4.2.7 State management. To implement the undo and redo functionalities requirements, a state management system has been implemented. This gives the user the ability to undo and redo actions done in your model, including but not limited to: resizing, adding elements, editing labels, and changing features about an edge. The main strategy in implementing this surrounds the takeSnapshot() function in the UndoRedo class, after the model is changed (e.g. label name is changed, edge is added, edge is changed), the model state is saved as a snapshot, and using the CTRL+Z and CTRL+Y shortcuts navigates us between these snapshots. This strategy works as the serialization has a very small file size, thus multiple of these snapshots can be easily stored without any storage or performance concerns.

4.2.8 Full keyboard shortcut support. Shortcut support is not natively supported in the reactFlow library, so they had to be manually implemented. SHortcuts for features of Deleting elements (DEL), saving files (CTRL+S), selecting all elements (CTRL+A), copying selected elements (CTRL+C), cutting selected elements (CTRL+C), pasting copied elements (CTRL+V), undo (CTRL+Z), and redo (CTRL+Y). These shortcuts were purposely chosen as they are commonplace amongst most computer tools, thus there would be an expectation from the user for them to be supported.

Additionally, this meant the user would not have to learn the shortcuts, and instead discover them naturally, lowering the time taken to learn the tool. Deletion created a unique bug, as when nodes were deleted, the IDs of nodes could potentially be duplicates, and this caused problems for Stephan's project, and occasionally some visual bugs. This meant that the management of ID's for nodes had to be rethought so that they were always unique.

4.2.9 Exporting Models. As requested by our experienced modellers, the ability to export our models as screenshots was added to the tool, an *Export* button triggers this feature.

4.2.10 Model Validation. The added temporal constraints can be confusing for beginners to understand, thus a model validation feature has been added to help correct users if they make an error in their model. The first way the validation feature works is by checking the user interface after every change made by the user, and using edge information to restrict what options the user has to change. This means that, rather than trying to throw an error if the user makes a mistake, the feature prevents the user from making a mistake in the first place. This can be seen in the way that the edge menu displays variables the user can change. These changeable variables are only available if they are relevant to the specific source and target nodes and type of lines.Table 4 is a summary table of the types of menu options, and the requirements needed for them to display. The second type of model validation displays errors if the types of edges the user has created are invalid in TREND. If an edge's connection is invalid, the edge will display as red, and selecting the edge will provide the exact error the error has made. These errors include using the incorrect edges between nodes. For example, temporal edges should only be made between two relationship types or entity types and subsumption edges can only be sourced at an entity or relationship type, and can only have an entity type as its target. After user testing, common errors that the beginner users made will be noted, and additional rules will be added to combat these common errors.

Table 4: Edge options and their requirements to be shown to the user.

Edge Option	**Requirements for rendering**
Edge Type	Always displayed
Optional	Edge is type temporal or atemporal
Cover	Edge is type subsumption
Edge Label	Edge is type temporal
Quantitative	Edge is type temporal and source and target nodes are both entities
(Quantitative) Value	Quantitative is set to True
Persistent	Edge is type temporal, target not is type relationship and source node is type entity
Cardinality (target node side)	Edge Type is atemporal
Cardinality (target node side)	Edge Type is atemporal, target not is type entity and source node is type relationship

4.2.11 Verbalization. When determining the preferred representation of temporal constraints in conceptual data models in [16], it was concluded that there was no clear favorite representation (when comparing graphical and textual representations), thus a multi-modal interface was suggested, to allow the switching between graphical notation and natural language. Since our tool is already graphical, this necessitates a feature which can represent our temporal constraints using natural language. To implement this feature, inspiration has been drawn from NORMA [8]. NORMA is a software tool that implements the Object-Role Modelling (ORM) language. The ORM schema are represented in a graphical notation, however, an additional feature called the ORM verbalization browser has been added, which represents the ORM schema in natural language to aid the user's understanding about the schema.

This feature has inspired the verbalization feature for this tool. When a temporal edge is selected, the bottom left of the screen displays the natural language representation of the edge. Additionally, when selecting a node, a natural language description of that node is also displayed. The feature works by using the type of edge, the source and target node, the variables attached to the edge, and the time quanta global variable. The specific natural language representation were taken from the supplementary material from [14]. For experienced modellers who find the feature distracting, this feature can also be disabled in the Leva control panel.

4.2.12 Other UI View-port Features. Other UI features were added to the canvas to assist modellers. This includes zoom controls, the ability to lock the canvas, fit-to-screen (this button will center the screen on your model), a mini-map in the bottom right hand corner to keep modellers orientated on the canvas. Additionally, the ability to lock the canvas, thus allowing the user to pan over it without accidentally moving something has been added too. Lastly, some global UI movement options have been added to the Leva control panel; snap-to-grid, pan-on-scroll, and zoom-on-double-click.

5 EVALUATION

The first aim from evaluation of our tool is to gain useful feedback from potential users of the tool to help improve the user experience and quality of the tool. The second aim was to determine the success of the tool from a user perspective, more specifically, to determine if the tool was more efficient and resulted in less errors than current solutions. These will both be evaluated in our user testing experiment. From a feature perspective, this tool was a success, as we were able to implement all the features we originally set out to achieve.

5.1 Unit and Performance Testing

To assist in achieving our non-functional requirements, unit tests were carried out to test the reliability of all functions in the tool. This would increase reliability as it would catch potential bugs that could damage the user experience. For example, these tests include the loading of incorrect models into the tool. A full list of the unit tests carried out (in-code) will be present in the final code submission.

Basic qualitative performance testing was also carried out, where over 100 nodes and edges were simultaneously loaded onto the canvas to test for bugs or slowdowns when handling a large amount of nodes. When doing this, no drop in frame-rate or bugs was experienced.

5.2 User Testing

5.2.1 Materials. This test required public open access to the TRENDy tool, so the tool was publicly hosted by the Vercel hosting platform, and can be accessed at https://trendy-blue.vercel.app/. The control tool chosen was Draw.io (formally diagrams.net), as this is the most popular tool for conceptual data modelling. In the feedback form, every tester reported that they had had previous experience with Draw.io. All users performed the tasks on their own laptop. As an added incentive for users to partake in user testing, pizza was also provided to users once completing the test. For each user test, a booklet with the following resources are required. A full example of this booklet is included in the appendix.

- **Consent form**: This must be signed before the user testing can begin. These would later be removed from the booklet and stored separately to keep user privacy.
- **Task based questionnaire**: The tasks take the form of three controlled natural language specifications of models that the users must build in their assigned tool. The models get progressively more difficult, and test all different types of features that are available in TREND, such as subsumption, dynamic extension and quantitative evolution. For example, the user would be given the following model description in controlled natural language:
  1. Employee is an entity type whose objects will always be an Employee.
  2. Each Manager is not a Manager for some time.
  3. An Employee may also become a Manager after 2 years.
  4. Each Manager was already an Employee for 2 years.

  From this, the user would be expected too add an atemporal entity labelled Employee, a temporal entity labelled Manager, and to connect these entities with temporal constraints (sourced from Employee and targeted at Manager), with both a quantitative (of 2 years), optional, further dynamic extension, and a quantitative (of 2 years), mandatory, past extension. A full list of these tasks, and their memorandum are in the appendix.
- **Feedback form**: This consists of a system usability scale (SUS), and for those users who were assigned to the TRENDy tool, additional feedback questions from asking about their experience using the tool to any suggestions they might have.
- **Resources explaining the TREND language**: These resources helped introduced the concepts added by TREND to the ER diagram. It includes an appendix table taken from [4] with all possible new temporal ER elements and an example of a full TREND model as well as its controlled natural language representation (also taken from [4]). The motivation of using these resources was both because of their brief yet thorough exploration of the topic, and because these resources were successfully used for the same purpose in previous TREND experiments.

5.2.2 Methods. To create the tasks the user's would be completing, the model would be created, and then the controlled natural language conversion (verbalization) would take place, and from this the sentences for the tasks would be used. This ensures a correct memo would be available to marks the user's results against (this is also included in the memo). User testers were recruited through the university communication channels and through snowball sampling techniques. They were incentivized to join with pizza as a reward once they had completed the evaluation tasks. The users are expected to already know the basics of ER conceptual data modelling. The following describes the protocol used for the user testing.

1. Participants arrive and are firstly asked to read through the consent form and sign it if they feel comfortable with participating in the test.
2. Participants are then randomly assigned a booklet with all the materials described in the previous section, the booklet they receive will determine which tool they use.
3. Participants are then told to read through the introduction to TREND, and also given a brief introduction verbally. They are also shown the TREND appendix and example as resources they can use to assist them.
4. After the participants are comfortable with their understanding of TREND, the timer is started and they must work through the three tasks outlined in the questionnaire.
5. Once they have completed the three tasks, the completion time is recorded, and they must send in the models that they have built along with their consent forms. Note that the consent forms are stored separately from everything else to keep participant anonymity.
6. Next participants must fill in the system usability scale.
7. If the participant was evaluating the TRENDy tool, then they must answer additional feedback questions, after the feedback is handed in, the experiment is over.
8. The results are then collected together, the models are marked on their correctness, and the data in recorded to be analysed.

The advantage of limiting the user testers to computer science students is that due to their education, they already have an understanding of ER modelling, and they would also form a large part of the potential users of this tool. It will also be noted that participants are not given an introduction to either tool in order to keep the experiment fair. They must learn the tool from exploring the user interface, as is done when users are introduced to most novel tools. Since all users had previous experience with draw.io, this could potentially put the newly developed tool at a disadvantage, however, this will better expose any unintuitive processes to my user interface effectively. The marking for the models will be continuous with experiment 11 in [4], where marks are allocated for each natural language sentence that is correctly represented on the graphical model. In this case, 4 marks were allocated for the first graph, 7 for the second model, and 10 for the third model.

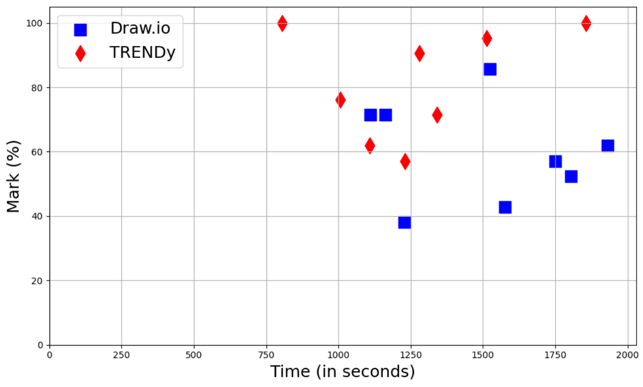
The system usability scale (SUS) [6] was used to evaluate the usability of the tool. It is a feedback metric that consists of 10 questions, each scored on a 5-point likert scale (ranging from "Strongly

Agree" to "Strongly Disagree"). These statements cover various aspects of usability, such as ease of use, complexity, and confidence using the system. The scores of these questions are added up and multiplied to be a score out of 100. In the evaluation, the SUS scores of the newly developed tool and the control tool will be compared. A full example of this scale is included in the questionnaire in the appendix. It is noted that a potential bias could have been present in the sampled demographic, where computer science students would give higher scores if they knew that the tool they used was built by another student. In an effort to mitigate this, the experiment was designed so that the modellers were not fully aware which part of the experiment they were testing for.

6 RESULTS

A total of 16 user tests were conducted, with 8 participants using the control tool (draw.io), and 8 participants using TRENDy. The full list of completed TREND models cannot be included due to length, however they will be included in the final code submission. For Draw.io, the control tool, the average mark was 12.63/21 (60.12%) and the average time was 25:10 to complete the tasks. For TRENDy, the average mark was 17.13/21 (81.55%) and the average time was 21:07 to complete the tasks. This means that when using TRENDy to design their conceptual data models, modellers could see a 16.01% reduction in time and a 21.43% improvement in their model correctness. The scatter plot below shows the results for each user evaluation that was conducted.

Figure 2: Scatter Plot showing Time and Marks for the task based evaluation.

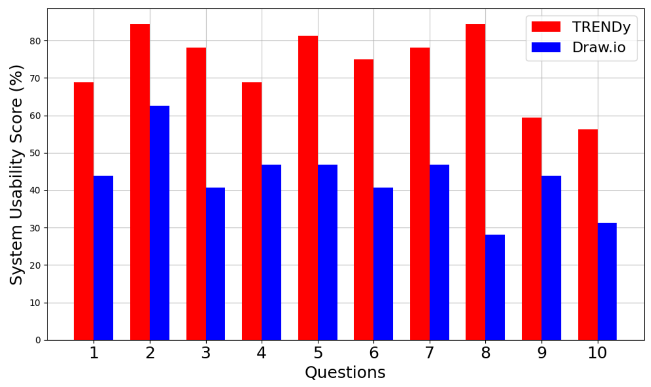


6.1 System Usability Scale

To calculate a SUS score, odd numbered questions give 4 points for "Strongly Agree", 3 points for "Somewhat Agree", 2 points for "Neutral", 1 point for "Somewhat Disagree", and 0 points for "Strongly Disagree". The opposite is true for even numbered questions. The 10 questions are scored and added for a total SUS score. The results in the tables represent an average score from all users. The full results tables are in the appendix.

(1) **I think I would like to use this tool frequently:** TRENDy: 68.75%, Draw.io: 43.75%

Figure 3: System Usability Scale Results



(2) **I found the tool unnecessarily complex:** TRENDy: 84.36%, Draw.io: 62.5%
(3) **I thought the tool was easy to use:** TRENDy: 78.26%, Draw.io: 40.63%
(4) **I think that I would need the support of a technical person to be able to use this system:** TRENDy: 68.76%, Draw.io: 46.88%
(5) **I found the various functions in this tool were well integrated:** TRENDy: 81.25%, Draw.io: 46.86%
(6) **I thought there was too much inconsistency in this tool:** TRENDy: 75%, Draw.io: 40.63%
(7) **I would imagine that most people would learn to use this tool very quickly:** TRENDy: 78.13%, Draw.io: 46.86%
(8) **I found the tool very cumbersome to use:** TRENDy: 84.36%, Draw.io: 28.13%
(9) **I felt very confident using the tool:** TRENDy: 59.38%, Draw.io: 43.75%
(10) **I needed to learn a lot of things before I could get going with this tool:** TRENDy: 56.25%, Draw.io: 43.75%

In general most of the mistakes were either made due to a misunderstanding on how the temporal constraints (commonly mixing up dynamic evolution and dynamic extension), or forgetting to include the clock icons when an object was temporal or frozen. TRENDy users saw a reduction in both of these types of mistakes, however they were prevalent in both sets of user testers. The average SUS amongst all questions for TRENDy was 73.4%, and 43.13

Note too that the TRENDy models look visually more coherent, as in Draw.io, users were adding random, visually incoherent icons and shapes to put together to make their models. Often colour, lines, and styles were inconsistent with each other (an example of this has been included in the appendix). Additionally, although not an offence that would warrant a mark reduction, many times in Draw.io models, the temporal constraints were made with stepped lines, rather than curved lines.

6.2 User Feedback

For the user feedback section, the following results were recorded. All completed feedback forms will be included in the final code submission.

6.2.1 *What tools have you used for conceptual data modelling in the past (i.e. for university assignment or work):*

Draw.io was mentioned in all 8 responses. Additional tools mentioned once were Miro, canva, and visio.

6.2.2 *Did you struggle with anything while using this tool? Please also point out any improvements you'd like to see implemented.*

- Bug: Optional and mandatory temporal constraints sometimes display incorrectly.
- Bug: Sometimes the edge option menu does not align properly.
- Bug: You cannot copy text, only figures.
- Bug: Nodes added to the canvas are automatically selected.
- Improvement: Add the ability to change arrow direction and icon type without deleting the node.
- Improvement: When an edge is selected, there should be feedback to that fact (e.g. the line is highlighted)
- Improvement: A brief helper panel or tutorial when first starting the tool.
- Improvement: Subsumption should be changed to Subsumption (ISA)

6.2.3 *Did the verbalization or graph validation features of this tool help you at any point while using the tool? Please explain.*

All 8 users reported that the verbalization feature helped them in understanding the temporal constraints, by providing clarity on what the constraints meant. There is 1 mention of the graph validation feature correcting a user that they had made a mistake.

6.2.4 *Is there any other feedback you would like to provide for the tool?*

Some of these suggestions are repeated from the second question (e.g. the optional and mandatory line bug and the request for a beginners' tutorial), new suggestions were adding a month time quanta, reordering the shapes to keep similar entities together, and changing the colour of the Leva control panel to make it look more integrated. The full set of results, including all completed TREND models and feedback forms will be included with the final code deliverable.

7 DISCUSSION

From the tasked based results, we can clear a clear improvement in the efficiency and correctness of modellers when using TRENDy over Draw.io. As noted in the results, the TRENDy models looked more visually coherent, due to the streamlined features over a generic modelling tool. The additional verbalization and graph verification feature did not entirely prevent mistakes from happening to TRENDy users, but it did reduce them. The nature of the mistakes was consistent throughout both tools, where incorrect temporal constraints and the failure to use the correct icons lost users the most marks. Not using the correct arrows (or not using any at all) in both subsumption relationships and temporal constraints was another common error.

From the system usability scale results, TRENDy consistently performed higher than Draw.io in the usability scores. This shows that the newly developed tool is more usable for its specific domain that generic modelling tools. The largest differences were observed in questions 3, 5, and 8. The fact users felt TRENDy to be easier to use, better integrated and less cumbersome is likely due to to more streamlined features that TRENDy offers that specifically cater to TREND, for example, the users would not have to sort through many different symbols and shapes to find a clock icon and rectangle to create a temporal entity, as they would in Draw.io. The smallest difference was observed in question 9, where users were only 15.63% more confident in using TRENDy over Draw.io. This is likely the difficulties of learning TREND that is prevalent in both tools.

From the user feedback questions, many useful insights were made, and some bugs were caught. All the suggestions made in the feedback have been implemented into the final tool. This will further improve the usability of the tool. The first questions further confirms that Draw.io was the most appropriate control tool to compare to TRENDy. The third questions confirms the success of our verbalization tool, as it was noted as being helpful all users. The graph validation tool was only recorded helping 1 out of the 8 users. This could be for a number of reasons, but could indicate that the rules were not thorough enough, so in the final tool, the list of validation rules will be expanded to catch more potential errors. Any errors made by TRENDy users will be analysed and if possible, additional rules will be made to combat these errors in the future.

8 CONCLUSIONS

This report has outlined the graphical tool that has been build to model in the TREND conceptual modelling language. The tool provides essential functionalities such as drag-and-drop modeling, schema generation, model validation, and support for TREND-specific elements like dynamic extension and evolution. The tool successfully implemented the features it set out to fulfill, and by integrating model validation and natural language verbalization, the tool simplifies the modeling process and reduces errors, lowering the barrier to entry for temporal data modeling while enhancing productivity for advanced users. From user testing, it was observed that TRENDy consistently beat a control tool in usability scores, and both reduced errors and increased efficiency for modellers. This can assist in increasing the reach and impact of temporal data modelling.

Future work for this tool includes implementing features that could not be implemented, such as notation conversion. Originally, there were plans to add the ability for modellers to easily convert between different popular ER notations (such as Chen's original notation, or the crow's feet notation), depending on which was their favorite. This feature was not included however, as TREND was not designed with other notations in mind (as it was designed graphically to be continuous with Chen's original ER notation), the look would not be graphically coherent with other ER notations. This feature could be implemented with different new graphical notations of TREND. Further testing should also be carried out on TRENDy to further refine its feature and to test a large enough sample size against control tools to obtain more statistically significant user evaluation results.

ACKNOWLEDGEMENTS

The author would like to thank all of the user testers for their valuable feedback and the supervisor and second reader for their suggestions and feedback which were vital in shaping this tool.

REFERENCES

[1] Alessandro Artale and Enrico Franconi. 1999. Temporal ER Modeling with Description Logics. In *Conceptual Modeling — ER '99*, Jacky Akoka, Mokrane Bouzeghoub, Isabelle Comyn-Wattiau, and Elisabeth Métais (Eds.). Springer, Berlin, Heidelberg, 81–95.

[2] Alessandro Artale, Christine Parent, and Stefano Spaccapietra. 2007. Evolving objects in temporal information systems. *Annals of Mathematics and Artificial Intelligence* 50, 1 (2007), 5–38.

[3] Andreas Behrend, Philip Schmiegelt, Jingquan Xie, Ronny Fehling, Adel Ghoneimy, Zhen Liu, Eric Chan, and Dieter Gawlick. 2015. Temporal State Management for Supporting the Real-Time Analysis of Clinical Data. *Advances in Intelligent Systems and Computing* 312 (01 2015), 159–170. https://doi.org/10.1007/978-3-319-10518-5_13

[4] Sonia Berman, C. Maria Keet, and Tamindran Shunmugam. 2024. The temporal conceptual data modelling language TREND. arXiv:2408.09427 [cs.DB] https://arxiv.org/abs/2408.09427

[5] Germán Braun, Christian Gimenez, Pablo Rubén Fillottrani, and Laura Cecchi. 2017. Towards conceptual modelling interoperability in a web tool for ontology engineering. In *III Simposio Argentino de Ontologías y sus Aplicaciones (SAOA)-JAIIO 46 (Córdoba, 2017)*.

[6] John Brooke. 1995. SUS: A quick and dirty usability scale. *Usability Evaluation in Industry* 189 (11 1995), 1–6.

[7] Yue Chen. 2022. Ontopanel: a diagrams.net plugin for graphical semantic modelling. In *Proceedings of the MSE 2022*. Darmstadt, Germany.

[8] Matthew Curland and Terry Halpin. 2007. Model Driven Development with NORMA. In *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. 286a–286a. https://doi.org/10.1109/HICSS.2007.384

[9] Ahmed El-abbassy. 2009. Tablet PC applications for education: database design using Hand-drawn ERD. *Journal of the ACS* 3 (2009), 1–13.

[10] Christian Gimenez, Germán Braun, Laura Cecchi, and Pablo Rubén Fillottrani. 2016. crowd: A Tool for Conceptual Modelling assisted by Automated Reasoning. In *II Simposio Argentino de Ontologías y sus Aplicaciones (SAOA 2016)-JAIIO 45 (Tres de Febrero, 2016)*.

[11] Heidi Gregersen. 2005. Timeer plus: A temporal eer model supporting schema changes. In *British National Conference on Databases*. Springer, Springer, Berlin, Heidelberg, 41–59.

[12] C.S. Jensen and R.T. Snodgrass. 1999. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering* 11, 1 (1999), 36–44. https://doi.org/10.1109/69.755613

[13] Christian S Jensen and Richard T Snodgrass. 2018. Temporal Database.

[14] Maria C Keet. 2017. Natural language template selection for temporal constraints. In *Proceedings of CREOL: Contextual Representation of Events and Objects in Language, Joint Ontology Workshops 2017* (Bolzano, Italy) *(CEUR Workshop Proceedings, Vol. 2017)*.

[15] Maria C Keet. 2023. Conceptual Data Models. In *The What and How of Modelling Information and Knowledge: From Mind Maps to Ontologies*. Springer, 49–79.

[16] Maria C Keet and Sonia Berman. 2017. Determining the preferred representation of temporal constraints in conceptual models. In *Conceptual Modeling: 36th International Conference, ER 2017, Valencia, Spain, November 6–9, 2017, Proceedings 36*. Springer, 437–450.

[17] Glenn E Krasner, Stephen T Pope, et al. 1988. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming* 1, 3 (1988), 26–49.

[18] Krishna Kulkarni and Jan-Eike Michels. 2012. Temporal features in SQL: 2011. *ACM Sigmod Record* 41, 3 (2012), 34–43.

[19] Thorsten Liebig, Marko Luther, Olaf Noppens, Mariano Rodriguez, Diego Calvanese, Michael Wessel, Matthew Horridge, Sean Bechhofer, Dmitry Tsarkov, and Evren Sirin. 2008. OWLlink: DIG for OWL 2.. In *OWLED*, Vol. 48.

[20] Ron McFadyen. 2015. AN ERD TOOL, Applied Computer Science, University of Winnipeg. *Submitted to an international journal* (2015).

[21] Deborah L McGuinness, Frank Van Harmelen, et al. 2004. OWL web ontology language overview. *World Wide Web Consortium (W3C) Recommendation* 10, 10 (2004), 70.

[22] C Theodoulidis, Pericles Loucopoulos, and Benkt Wangler. 1991. A conceptual modelling formalism for temporal database applications. *Information Systems* 16, 4 (1991), 401–416.

[23] Andrew Zimdars, David Maxwell Chickering, and Christopher Meek. 2013. Using temporal data for making recommendations. *arXiv preprint arXiv:1301.2320* (2013).

9 APPENDIX

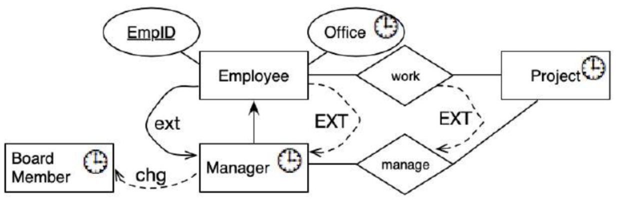
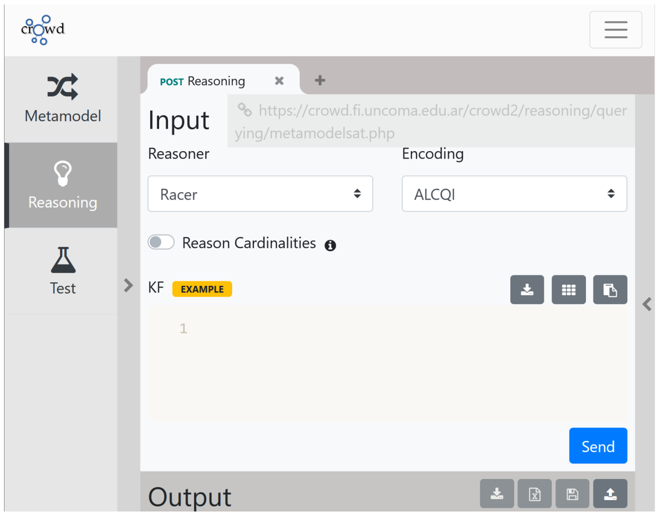


Figure 4: ER diagram extended by TREND. Taken from Figure 8 of [4].

Temporal elements, such as an employee's office, are indicated with the clock icon to reflect their potential for change over time. For example, an employee might eventually become a manager, and to qualify as a manager, the individual must have been an employee (i.e., a manager remains an instance of an employee). This relationship is represented by mandatory past and optional future dynamic extensions. Additionally, a manager may later optionally become a Board Member, which is not an instance of a manager, representing a future optional dynamic evolution. This is displayed after selecting "Entity Relationship", notice the inconsistency, where what is being displayed is clearly not an Entity Relationship diagram, and is more akin to a UML diagram.

Figure 5: Crowd's Reasoning feature. Screenshot taken from http://crowd.fi.uncoma.edu.ar:3335/api.



Input https://crowd.fi.uncoma.edu.ar/crowd2/reasoning/querying/metamodelsat.php

Reasoner: Racer

Encoding: ALCQI

Reason Cardinalities

Figure 6: Crowd's user interface, using EER. Screenshot taken from http://crowd.fi.uncoma.edu.ar:3335/editor/eer.

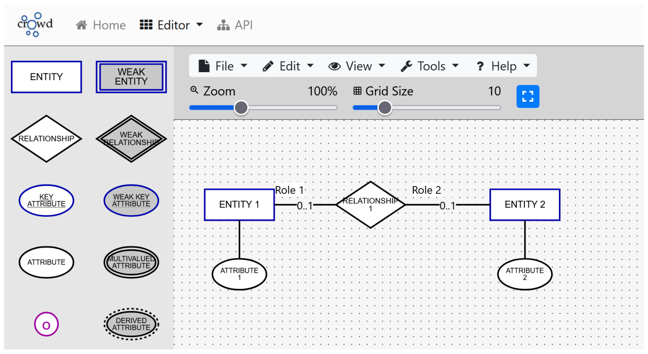


Figure 7: Draw.io's user interface, using ER. Screenshot taken from draw.io native: https://github.com/jgraph/drawio-desktop.

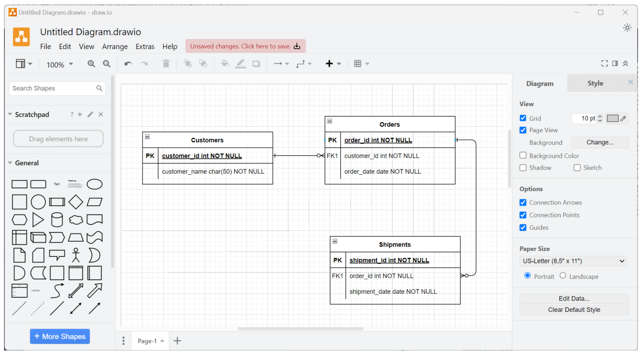
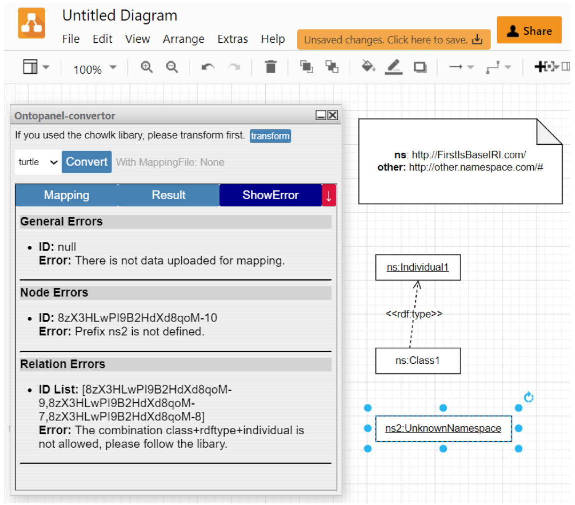

Untitled Diagram.drawio
File Edit View Arrange Extras Help
Unsaved changes. Click here to save.
Customers
PK customer_id int NOT NULL
customer_name char(50) NOT NULL
Orders
PK order_id int NOT NULL
FK1 customer_id int NOT NULL
order_date date NOT NULL
Shipments
PK shipment_id int NOT NULL
FK1 order_id int NOT NULL
shipment_date date NOT NULL
Diagram
Style
+ More Shapes
Page-1

Figure 8: Ontopanel's reasoner working in Draw.io. Taken from [7].



If you used the chowlk libary, please transform first.

ns: http://FirstIsBaseIRI.com/
other: http://other.namespace.com/#

General Errors

- **ID:** null
  Error: There is not data uploaded for mapping.

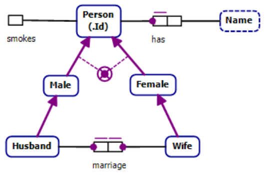
Node Errors

- **ID:** 8zX3HLwPI9B2HdXd8qoM-10
  Error: Prefix ns2 is not defined.

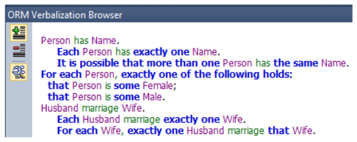
Relation Errors

- **ID List:** [8zX3HLwPI9B2HdXd8qoM-9,8zX3HLwPI9B2HdXd8qoM-7,8zX3HLwPI9B2HdXd8qoM-8]
  Error: The combination class+rdftype+individual is not allowed, please follow the libary.

Figure 9: The ORM language represented in graphical and textual forms in NORMA. Taken from [8].



(a) *ORM schema*.



Person has Name.
    Each Person has **exactly one** Name.
    It is possible that more than one Person has **the same** Name.
For each Person, **exactly one of the following holds:**
  that Person is **some** Female;
  that Person is **some** Male.
Husband marriage Wife.
    Each Husband marriage **exactly one** Wife.
    For each Wife, **exactly one** Husband marriage **that** Wife.

(b) *ORM schema verbalization*.

9.1 Full System Usability Scale Results

Table 5: Survey Results for Draw.io Tool Usage

Question	Strongly Disagree (Count)	Somewhat Disagree (Count)	Neutral (Count)	Somewhat Agree (Count)	Strongly Agree (Count)	Total (Score)
I think I would like to use this tool frequently.	1	2	3	2	0	14
I found the tool unnecessarily complex.	0	5	2	1	0	20
I thought the tool was easy to use.	2	1	3	2	0	13
I think I would need the support of a technical person to use this system.	1	1	2	4	0	15
I found the various functions in this tool were well integrated.	0	3	3	2	0	15
I thought there was too much inconsistency in this tool.	0	3	1	2	2	13
I would imagine that most people would learn to use this tool very quickly.	1	1	4	2	0	15
I found the tool very cumbersome to use.	0	0	1	7	0	9
I felt very confident using the tool.	2	1	3	1	1	14

Table 6: Survey Results for Draw.io Tool Usage

Question	Strongly Disagree (Count)	Somewhat Disagree (Count)	Neutral (Count)	Somewhat Agree (Count)	Strongly Agree (Count)	Total (Score)
I think I would like to use this tool frequently.	1	2	3	2	0	14
I found the tool unnecessarily complex.	0	5	2	1	0	20
I thought the tool was easy to use.	2	1	3	2	0	13
I think I would need the support of a technical person to use this system.	1	1	2	4	0	15
I found the various functions in this tool were well integrated.	0	3	3	2	0	15
I thought there was too much inconsistency in this tool.	0	3	1	2	2	13
I would imagine that most people would learn to use this tool very quickly.	1	1	4	2	0	15
I found the tool very cumbersome to use.	0	0	1	7	0	9
I felt very confident using the tool.	2	1	3	1	1	14

9.2 Full Serialization Example (the first model from the memorandum)

```
  {
"nodes": [
  {
    "id": "1X5cPI",
    "type": "shape",
    "position": {
      "x": 400,
      "y": 240
    },
    "data": {
      "type": "atemporalEntity",
      "label": "Employee",
      "identifier": false,
      "disjoint": false
    },
    "style": {
      "width": 120,
      "height": 80
    },
    "selected": false,
    "measured": {
      "width": 120,
      "height": 80
    },
    "dragging": false
  },
  {
    "id": "2sPb8z",
    "type": "shape",
    "position": {
      "x": 540,
      "y": 100
    },
    "data": {
      "type": "temporalEntity",
      "label": "Manager",
      "identifier": false,
      "disjoint": false
    },
    "style": {
      "width": 120,
      "height": 80
    },
    "selected": false,
    "measured": {
      "width": 120,
      "height": 80
    }
  }
],
"edges": [
  {
    "type": "temporalEdge",
    "style": {
      "stroke": "black",
      "strokeWidth": 2
    },
    "source": "1X5cPI",
    "sourceHandle": "top",
    "target": "2sPb8z",
    "targetHandle": "left",
    "id": "edge-1725822628954",
    "data": {
      "label": "EXT",
      "quantitative": true,
      "value": "2",
      "optional": "Optional"
    }
  },
  {
    "type": "temporalEdge",
    "style": {
      "stroke": "black",
      "strokeWidth": 2
    },
    "source": "1X5cPI",
    "sourceHandle": "right",
    "target": "2sPb8z",
    "targetHandle": "bottom",
    "id": "edge-1725822630290",
    "data": {
      "label": "ext",
      "optional": "Mandatory",
      "quantitative": true,
      "value": "2"
    }
  }
 ]
}
```

9.3 Memorandum for the Task-based evaluation

On the following page, the expected outputs for the three models from the evaluation are shown

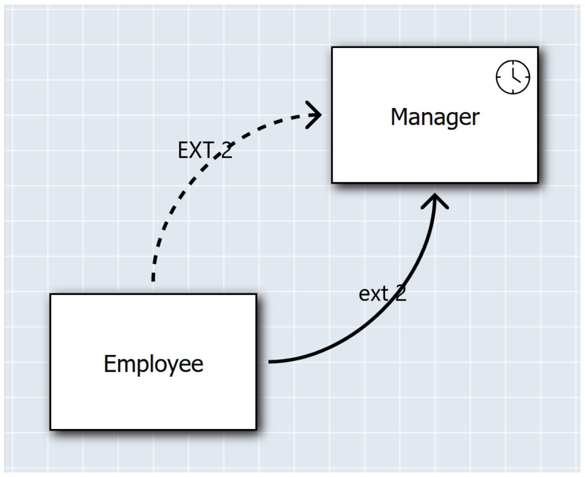


Figure 10: Memo: Task 1

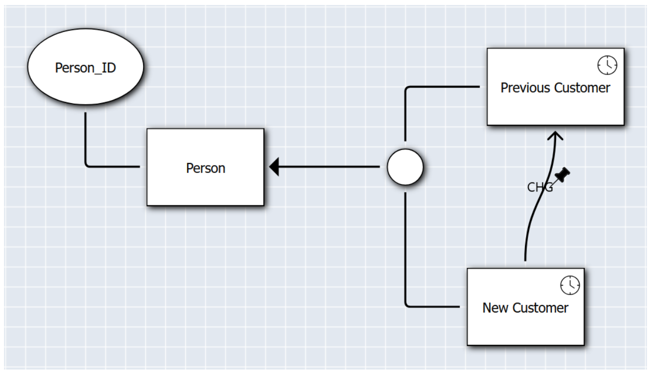


Figure 11: Memo: Task 2

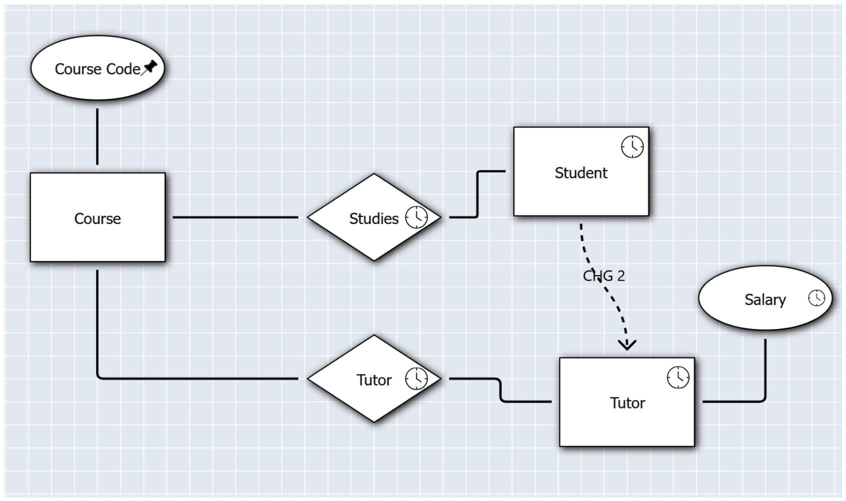


Figure 12: Memo: Task 3

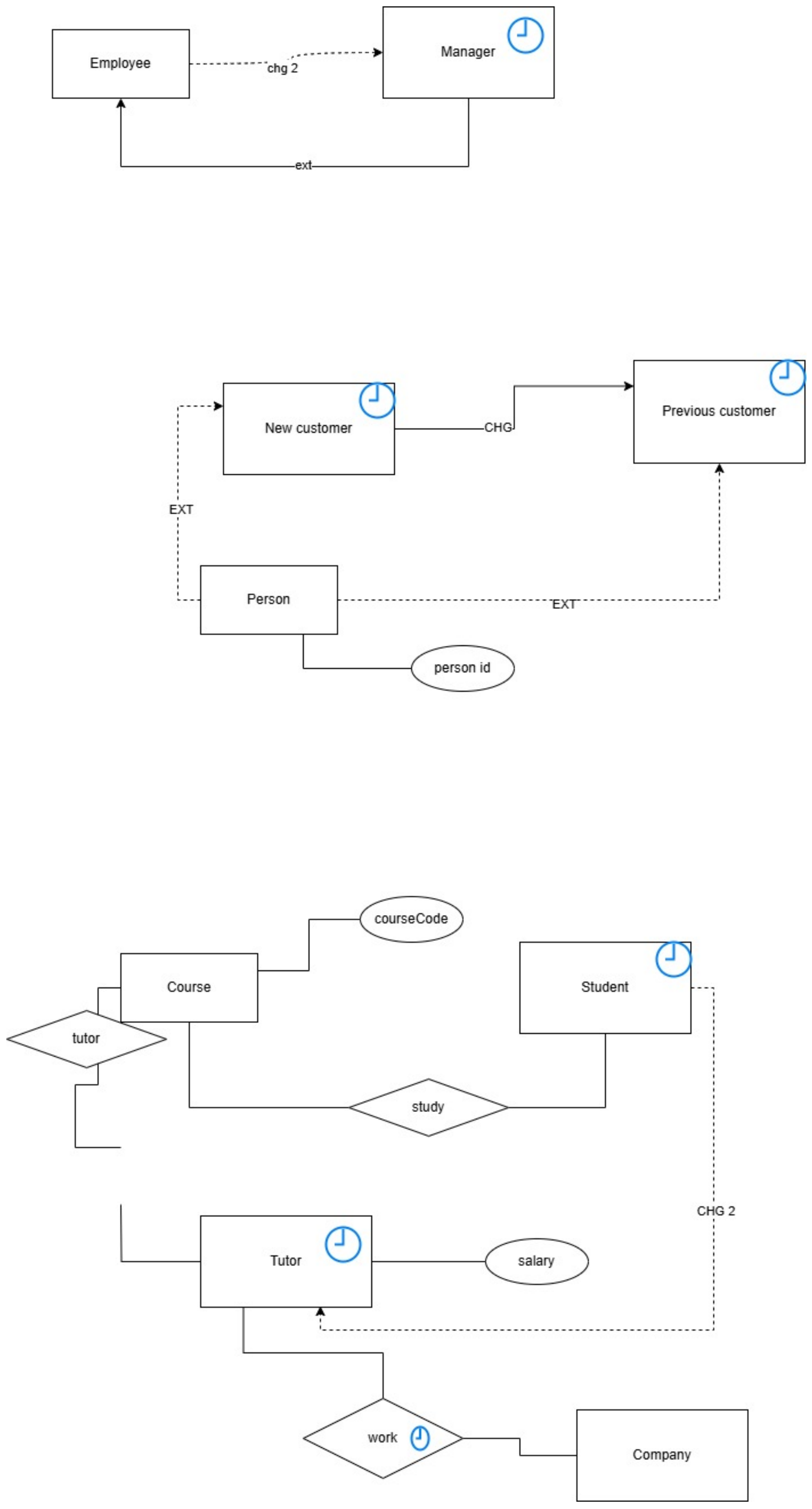


Figure 13: An example of the visual inconsistencies found when using Draw.io

9.4 Booklet given to students

The following pages form part of the booklet given to students to complete their task based evaluation. It includes the consent form, questionnaire, and supplementary TREND resources to assist users in their understanding of temporal conceptual data modelling (A brief introduction to trend, followed by the graphical model of Assignment 5 (k) and an appendix of all TREND features.).

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF CAPE TOWN
PRIVATE BAG X3
RONDEBOSCH 7701
SOUTH AFRICA

RESEARCHER/S: Richard Taylor
TELEPHONE: +27-79-693478
E-MAIL: Tylric007@myuct.ac.za

Informed Voluntary Consent to Participate in Research Study

Project Title: Facilitating the implementation of the Trend Conceptual Data Modelling Language

Invitation to participate, and benefits: You are invited to participate in a research study conducted with computer science students. The study aim is to develop a tool that allows the user to design and implement a model based on the TREND Temporal Conceptual Data Modelling Language. There are currently no tools that are designed for Temporal Conceptual Data Modelling, and there exists no software that generates temporal database schemas based on a model. The project aims to address these problems by building a modelling tool, as well as designing and implementing an algorithm to generate a database schema based on the user's model. This is expected to lower the barrier of entry for temporal modelling, as well as increase productivity for experienced modellers. I believe that your experience would be a valuable source of information, and hope that by participating you may gain useful knowledge.

Procedures: During this study, you will be asked to complete task-based tests in the form of questions that will be provided to you. These questions will outline model specifications in controlled natural language, and you must use these specifications to create conceptual data models using a randomly assigned tool. During these tests, any mistakes that you make will be recorded, as well as the time it took to create the model. It will be noted how fast you can model and how few mistakes you make.

Disclaimer/Withdrawal: Your participation is completely voluntary; you may refuse to participate, and you may withdraw at any time without having to state a reason and without any prejudice or penalty against you. Should you choose to withdraw, the researcher commits not to use any of the information you have provided without your signed consent. Note that the researcher may also withdraw you from the study at any time.

Confidentiality: All information collected in this study will be kept private in that you will not be identified by name or by affiliation to an institution. Confidentiality and anonymity will be maintained as pseudonyms will be used.

What signing this form means: By signing this consent form, you agree to participate in this research study. The aim, procedures to be used, as well as the potential risks and benefits of your participation have been explained verbally to you in detail, using this form. Refusal to participate in or withdrawal from this study at any time will have no effect on you in any way. You are free to contact me, to ask questions or request further information, at any time during this research.

I agree to participate in this research (tick one box) ☐ Yes ☐ No __________ (Initials)

______________________	______________________	______
Name of Participant	Signature of Participant	Date
______________________	______________________	______
Name of Researcher	Signature of Researcher	Date

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF CAPE TOWN
PRIVATE BAG X3
RONDEBOSCH 7701
SOUTH AFRICA

RESEARCHER/S: Richard Taylor
TELEPHONE: +27-79-693478
E-MAIL: Tylric007@myuct.ac.za
URL: https://sit.uct.ac.za/department-computer-science

User Testing and Feedback Questionnaire

Project Title: Facilitating the implementation of the Trend Temporal Conceptual Data Modelling Language

1. Please make sure you have signed the Informed Voluntary Consent form.

2. Read the introduction to TREND insert to get an understanding of the basics of the TREND temporal transition constraints.

3. Once you are comfortable with your understanding of TREND, please can you complete the following tasks, using the tool Draw.io (**app.diagrams.net).**

   If the users are using TRENDy, …the tool TRENDy (**trendy-blue.vercel.app)** is displayed instead.

Modelling

Consider the following description of a small temporal model. Try to model it with TREND using the tool assigned. Once you have completed the model, you must export the model as *usernumber_task_x.png*.

I. Modelling your first temporal relationship between an employee and a manager.

Employee is an entity type whose objects will always be an Employee.
Each Manager is not a Manager for some time.
An Employee may also become a Manager after 2 years.
Each Manager was already an Employee for 2 years.

II. Modelling the temporal relationship of a customer in a company.

Person is an entity type whose objects will always be a Person.
Each object in entity type Person having attribute Person_ID has Person_ID at all times.
Each New-Customer is not a New-Customer for some time.
Each Previous-Customer is not a Previous-Customer for some time.
New-Customer is a Person.
Previous-Customer is a Person.
Each New-Customer must evolve to a Previous-Customer ceasing to be a New-Customer, and this remains so indefinitely.

III. Modelling a university temporal database.

Course is an entity type whose objects will always be a Course.
Once the value for Course Code is set, it cannot change anymore.
Each Student is not a Student for some time.
A Student Studies a course.
The objects participating in a fact in {A Student Studies a course} do not relate through {Studies} at some time.
Each Tutor is not a Tutor for some time.
A Tutor Tutors a Course.
Each object in entity type Tutor having attribute Salary does not have a Salary at some time
The objects participating in a fact in { A Tutor Tutors a Course} do not relate through {Tutors} at some time.
A Student may evolve to become a Tutor after 2 years, ceasing to be a Student.

Feedback

System Usability Scale:

	Strongly Disagree	**Somewhat Disagree**	**Neutral**	**Somewhat Agree**	**Strongly Agree**
I think I would like to use this tool frequently.	☐	☐	☐	☐	☐
I found the tool unnecessarily complex.	☐	☐	☐	☐	☐
I thought the tool was easy to use.	☐	☐	☐	☐	☐
I think that I would need the support of a technical person to be able to use this system.	☐	☐	☐	☐	☐
I found the various functions in this tool were well integrated.	☐	☐	☐	☐	☐
I thought there was too much inconsistency in this tool	☐	☐	☐	☐	☐
I would imagine that most people would learn to use this tool very quickly.	☐	☐	☐	☐	☐
I found the tool very cumbersome to use.	☐	☐	☐	☐	☐
I felt very confident using the tool.	☐	☐	☐	☐	☐
I needed to learn a lot of things before I could get going with this tool.	☐	☐	☐	☐	☐

For those participants who used the *TRENDy* tool: [This is only shown to TRENDy users]

1. What tools have you used for conceptual data modelling in the past (i.e. for university assignment or work):

____

____

2. Did you struggle with anything while using this tool? Please also point out any improvements you'd like to see implemented:

____

____

____

____

____

____

____

____

3. Did the verbalization or graph verification features of this tool help you at any point while using the tool? Please explain:

____

____

____

____

4. Is there any other feedback you would like to provide for the tool?

____

____

____

____

____

____

A Brief Introduction to Temporal Modelling with TREND

1. Introduction

Modelling of temporal constraints for information systems has received attention since the mid-1990s (e.g., [1,2,3]), as it adds expressiveness to ensure data integrity. For instance, to ensure each Graduate must have been a Student at that university before (an *evolving object*), that a couple registered as divorcing in a census database must have been marrying before (an *evolving relation*), or that workers may not always have an Office (*temporal attribute*). The key things that can happen over time can be depicted graphically along a timeline as shown in Fig.1. The horizontal lines and dots depict an object's life—for a duration or an instant—as an instance of an entity type, a tuple as a member of a relationship, or an attribute. The 'snapshot' is also called 'rigid' and means that that object is always an instance of the entity type for the entirety of its lifetime, like an apple is always an apple, and you will always be a human for your entire life. For temporal entities, they are an instance of something for some time only, as indicated with the 'temporal options' (also called 'antirigid'). For instance, you as instance of Human are an instance of Employee only for some time of your life, and the attribute of a country's COVID-19 Lockdown level is temporary as well (it didn't exist 2 years ago and one may hope we won't have it anymore in 2 years' time). Also, things can happen to the instances over time, i.e., a transition may take place in the future, or have taken place in the past: some object k as an instance of type A may also in the future become (or in the past have been) an instance of type B for some time. Such temporal notions are typically considered from a reference point, which, for conceptual modelling is normally the 'now' that also has a past and a future.

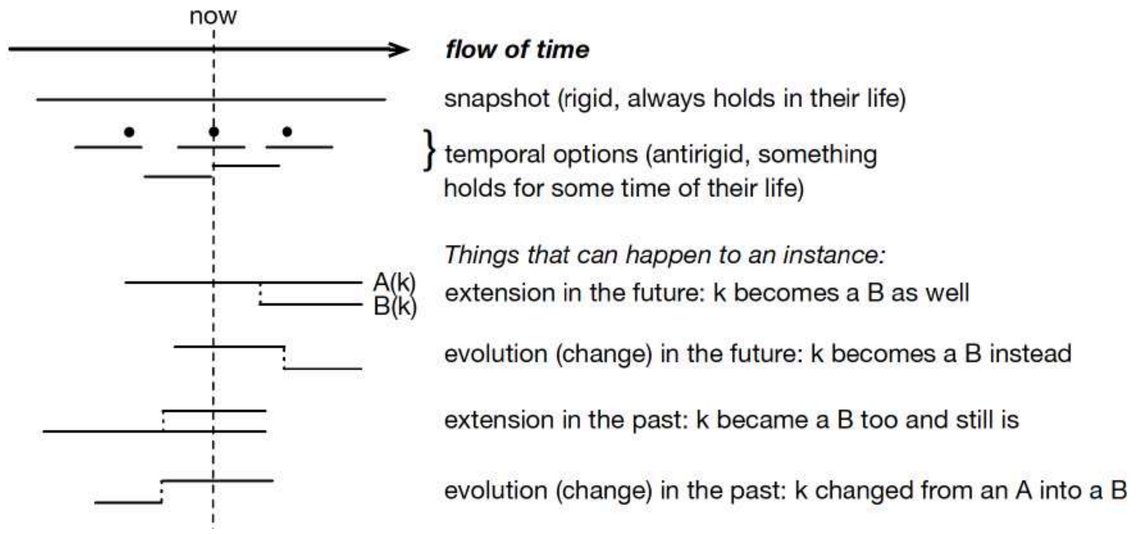

now
flow of time
snapshot (rigid, always holds in their life)
temporal options (antirigid, something holds for some time of their life)
Things that can happen to an instance:
A(k)
B(k)
extension in the future: k becomes a B as well
evolution (change) in the future: k becomes a B instead
extension in the past: k became a B too and still is
evolution (change) in the past: k changed from an A into a B

Figure 1. Graphical depiction of some key temporal aspects of an entity's life (first as A, then somehow B).

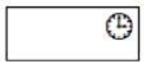
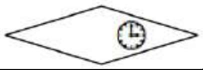
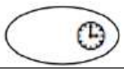
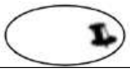
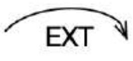
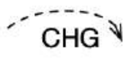
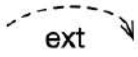
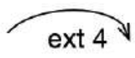
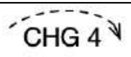
Also in Big Data and the Internet of Things, data is temporal, and thus requiring a temporal data model. For this assignment we will use an extended version of ER_{VT} [4] called TREND (**T**emporal information **R**epresentation in **En**tity-Relationship **D**iagrams) [5,6].

2. The TREND conceptual data modelling language

TREND focusses on temporalising entity types, relationships and attributes, and specifying temporal transition constraints. The core transition constraints are *dynamic extension (EXT)* and *dynamic evolution* that *changes (CHG)* things. In an extension, the entity is *also* an instance of the other entity whereas with the change, the entity *ceases* to be an instance of the source entity type. An example of extension is when an Employee becomes a Manager, but they still remain an Employee. An example of change is when a SeniorLecturer is promoted to the rank of AssociateProfessor, since then they are no longer a SeniorLecturer anymore. This temporal information cannot be represented in 'plain' atemporal EER that you have learned in class, but we need new elements to indicate that additional information. Those adornments have to be relatively easy for the analyst to use when modelling and easy to understand for the domain expert. After multiple iterations of design and evaluation, the following additions have been made, which are summarised in Table 1 below.

A clock icon indicates a temporal element, and arrows labelled with CHG and EXT represent temporal transitions in the *future*. Labels are chg and ext when the transition constraints refers to the *past*, e.g. a Graduate must have been a Student before. To remember the past/future difference, one might think of it as 'the past has already gone down, lowercase' and 'the future is a step up, uppercase'. Both future and past transitions are needed when these transition constraints differ: e.g., Student becoming a Graduate in the future is optional, but Graduate having been a Student in the past is mandatory (required/compulsory). A dashed arrow shaft denotes an optional transition and a solid shaft denotes a mandatory transition. Any transition constraint can include a time quantity to indicate how much time must elapse before the transition can occur. An attribute with a drawing pin means that the attribute is *frozen*, i.e., is not allowed to change anymore.

Table 1. Selection of the notation of the TREND diagram language.

Icon	**Name**	**Description**
	Temporal entity type	Entities of this type are ***not always*** entities of this type
	Temporal relationship	Relations of this relationship are ***not always*** relations of this type
	Temporal attribute	The entity does ***not always*** have a value for this attribute
	Frozen attribute	Once set, retains (drawing pin) its value in perpetuity (***forever***)
	Mandatory dynamic extension in the future	***Must*** (solid shaft) in the future (uppercase) ***extend to*** (EXT) also be an instance of the target type
	Optional dynamic evolution in the future	***May*** (dashed shaft) in future (uppercase) ***change*** (CHG) instead into the target type
	Optional dynamic extension in the past	***May*** in the past (lowercase) ***have extended*** (ext) from also being a member of the source type
	Mandatory quantitative dynamic extension in the past	***Must*** in the past (lowercase) ***have extended*** (ext) from also being of the source type ***4*** time units earlier
	Optional quantitative dynamic evolution in future	***May*** in the future ***change*** (CHG) instead to the target type after ***4*** time units

3. Examples of TREND models

An example TREND diagram is shown in Figure 2. Office is a temporal attribute, as, over their period of employment, employees may have their own office at some times and not have their own office at other times. The mandatory transition ext indicates that a manager must have been working for the company as a regular employee before being promoted to manager, and thus that that transition from employee to manager happened in the past. Not all employees will be promoted to manager, hence, the optional EXT from employee to manager. Likewise, the transition from work to manage is optional.

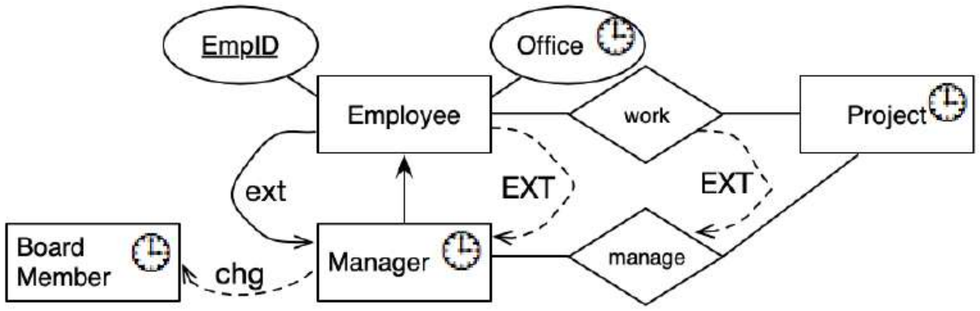


Figure 2. Example of a temporally extended ER diagram using TREND.

Figure 3 shows three examples of temporal entities and changes in the past. The clock icons indicate that an undergraduate is not always an undergraduate, and similarly for postgraduate, tadpole, frog, lawyer and articled clerk (every lawyer has to work as an articled clerk for 2 years before they can be a lawyer.) The arrows all have solid shafts indicating every frog must have been a tadpole before, every postgrad must have been an undergrad before, every lawyer must have been an articled clerk beforehand. Each diagram could have a CHG arrow added with a dashed shaft, as evolution in the future is not guaranteed for any of these cases: tadpoles can die, and many undergraduates and articled clerks fail to advance to postgraduate (or lawyer) status.

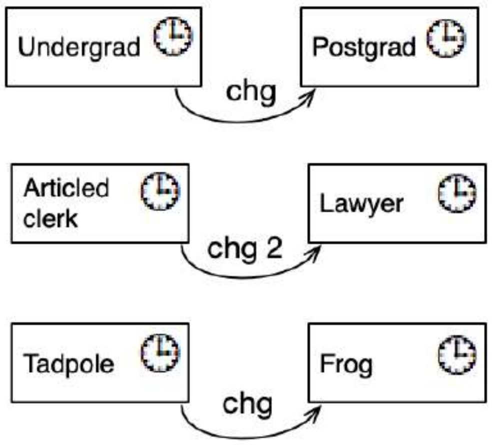


Figure 3. Examples of DEV$^-$ transitions using TREND.

Figure 4 is given to see if you can understand the notation without relying on common knowledge of the real world. Here M, P and S are entity types. If an entity is of type P at any point in time then it is always of type P. If an entity is of type S at any point in time then it is always of type S. If an entity is of type M at any point in time then it may not be of type M at some other time. An M entity may have an attribute N. If an M entity has an attribute N at any point in time it may sometimes not have an attribute N i.e. it may not always have an attribute N. A P entity may also become an M at some time and still remain a P. Every M entity must also have been a P at some time and still is a P. Q and R are 2 relationships that may hold at times between S entities and P entities. If a P and an S are related through R at any point in time they may not always be related through R. If a P and an S are related through Q at any point in time they may not always be related through Q. If a P and an S are related through Q at any point in time then they may have been related through R at another point in time and if so that relationship has to have been 2 time units beforehand.

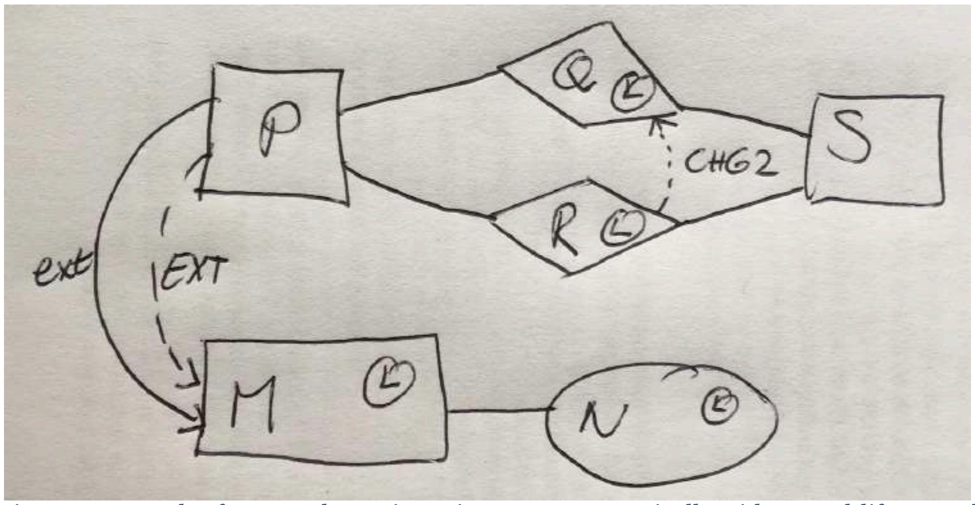
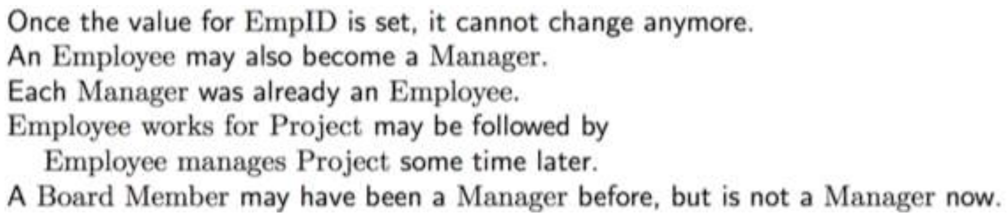


Figure 4. Example of temporal notation using TREND, generically without real-life examples.

4. Talking about temporal models and TREND

One may prefer visualisations of the information, but it also may be useful to be able to talk about the temporal information in a systematic way and some domain experts do prefer text over diagrams. Either way, generating natural language sentences from a conceptual model can be done automatically with a *controlled natural language* in software that then outputs natural language sentences. For instance, the dynamic extension among entity types can be captured with a template 'A(n) …. may also become a(n) …', so that with the sample model in Figure 2, it will fetch the elements of the EER diagram to generate sentences; in this case, it will generate the sentence 'An Employee may also become a Manager'. More examples are included in Figure 5.



Once the value for EmpID is set, it cannot change anymore.
An Employee may also become a Manager.
Each Manager was already an Employee.
Employee works for Project may be followed by
Employee manages Project some time later.
A Board Member may have been a Manager before, but is not a Manager now.

Figure 5. Examples of sentences generated from the sample diagram in Figure 2, using a controlled natural language for TREND.

References

1. Gianni, D., Bocciarelli, P., D'Ambrogio, A.: Temporal capabilities in support of conceptual process modeling using object-role modeling. In: Proc. of DEVS Inte- grative'14 (2014)
2. Keet, C.M., Ongoma, E.A.N.: Temporal attributes: their status and subsumption. In: Proc. of APCCM'15. CRPIT, vol. 165, pp. 61–70. CRPIT (2015)
3. Khatri, V., Ram, S., Snodgrass, R.T., Terenziani, P.: Capturing telic/atelic tem- poral data semantics: Generalizing conventional conceptual models. Transactions on Knowledge and Data Engineering 26(3), 528–548 (2014)
4. Artale, A., Parent, C., Spaccapietra, S.: Evolving objects in temporal information systems. Annals of Mathematics and Artificial Intelligence 50(1-2), 5–38 (2007)
5. Keet, C.M., Berman, S. Determining the preferred representation of temporal constraints in conceptual models. 36th International Conference on Conceptual Modeling (ER'17). Mayr, H.C., Guizzardi, G., Ma, H., Pastor., O. (Eds.). Springer LNCS vol. 10650, 437-450. 6-9 Nov 2017, Valencia, Spain.
6. Keet, C.M. Natural language template selection for temporal constraints. CREOL: Contextual Representation of Events and Objects in Language, Joint Ontology Workshops 2017, 21-23 September 2017, Bolzano, Italy. CEUR-WS Vol. 2050. 12p.

Assignment 5 question (k)

Draw a TREND data model for the following description:

Flight is an entity type whose objects will always be a flight.

Client is an entity type whose objects will always be a client.

Traveller is a client.

Previous-customer is a client.

VIP-customer is a client.

Each traveller is not a traveller for some time.

Each previous-customer is not a previous-customer for some time.

Each VIP-customer is not a VIP-customer for some time.

A client may also become a traveller.

A client may also become a VIP-customer.

Each traveller must evolve to a previous-customer ceasing to be a traveller.

A previous-customer may also become a traveller.

A previous-customer may also become a VIP-customer.

Each VIP-customer was already a previous-customer.

Each VIP-customer was already a traveller.

A traveller books a flight

A traveller pays for a flight.

A previous-customer took a flight.

Each traveller books a flight will be followed by traveller pays for flight after exactly 3 days, terminating the traveller books a flight relation.

Each previous-customer took a flight must have been preceded by traveller pays for flight, and terminating that traveller pays for flight relation.

Each object in entity type client having attribute company does not have a company at some time.

Each object in entity type flight having attribute delay has delay at all times.

Once the value for arrival time is set, it cannot change anymore.

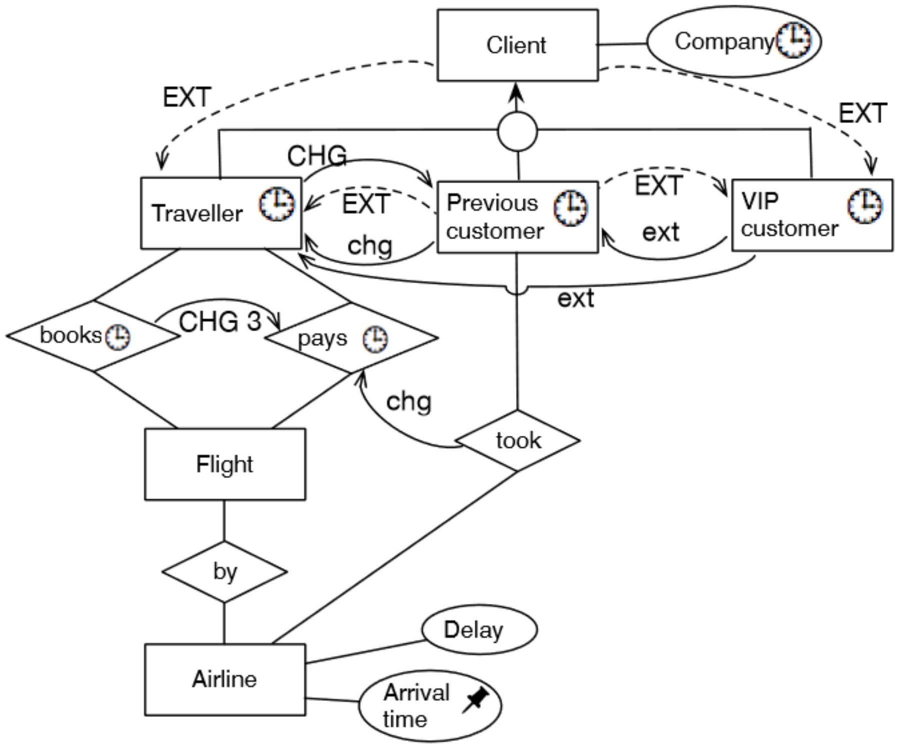
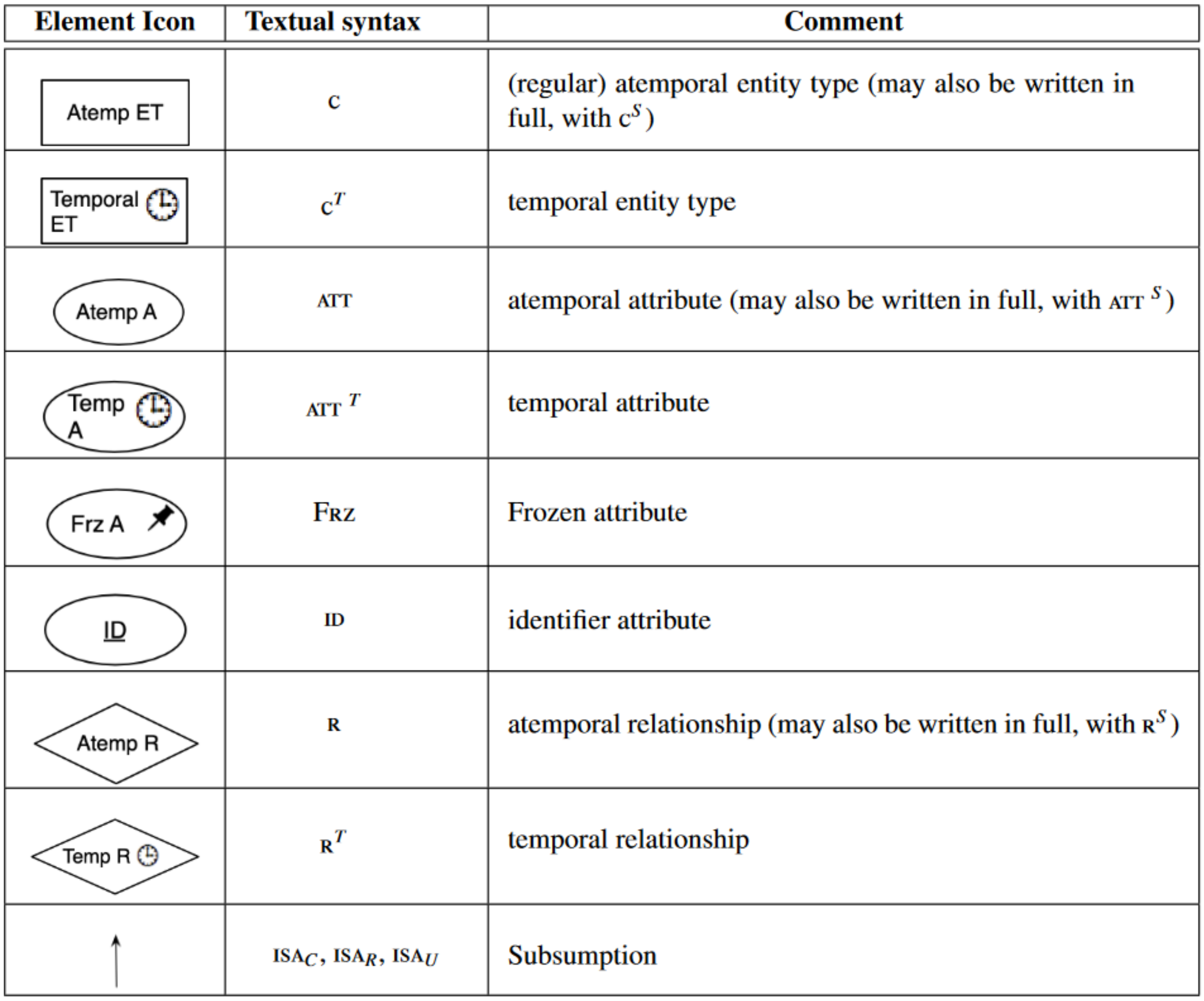
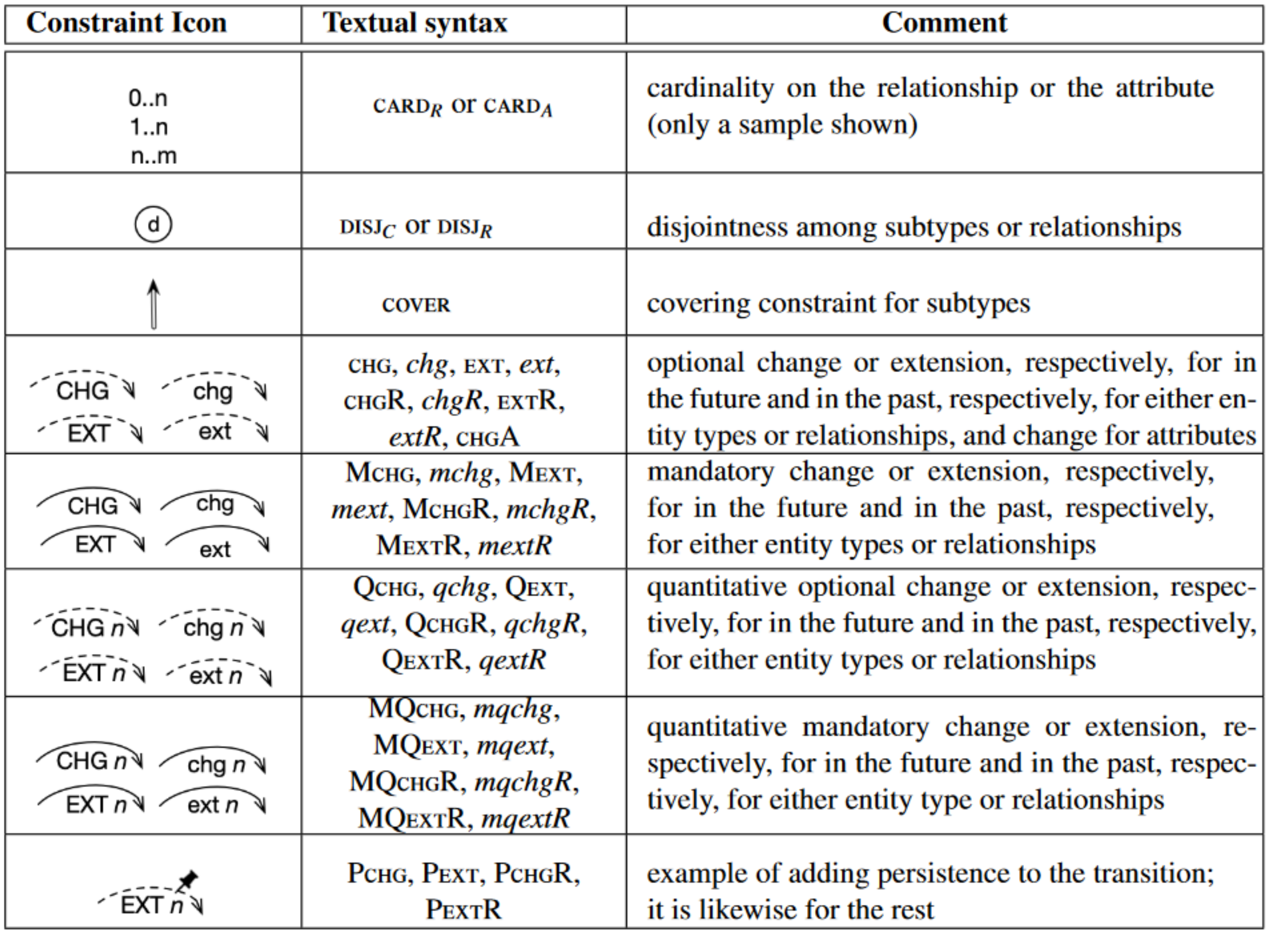


Figure 14: Model representation of Assignment 5 question (k)



Element Icon	Textual syntax	Comment
Atemp ET	C	(regular) atemporal entity type (may also be written in full, with C^S)
Temporal ET	C^T	temporal entity type
Atemp A	ATT	atemporal attribute (may also be written in full, with ATT^S)
Temp A	ATT^T	temporal attribute
Frz A	FRZ	Frozen attribute
ID	ID	identifier attribute
Atemp R	R	atemporal relationship (may also be written in full, with R^S)
Temp R	R^T	temporal relationship
↑	ISA_C, ISA_R, ISA_U	Subsumption

Figure 15: TREND Appendix A



Constraint Icon	Textual syntax	Comment
0..n 1..n n..m	CARD_R or CARD_A	cardinality on the relationship or the attribute (only a sample shown)
d	DISJ_C or DISJ_R	disjointness among subtypes or relationships
↑	COVER	covering constraint for subtypes
CHG chg EXT ext	CHG, *chg*, EXT, *ext*, CHGR, *chgR*, EXTR, *extR*, CHGA	optional change or extension, respectively, for in the future and in the past, respectively, for either entity types or relationships, and change for attributes
CHG chg EXT ext	MCHG, *mchg*, MEXT, *mext*, MCHGR, *mchgR*, MEXTR, *mextR*	mandatory change or extension, respectively, for in the future and in the past, respectively, for either entity types or relationships
CHG *n* chg *n* EXT *n* ext *n*	QCHG, *qchg*, QEXT, *qext*, QCHGR, *qchgR*, QEXTR, *qextR*	quantitative optional change or extension, respectively, for in the future and in the past, respectively, for either entity types or relationships
CHG *n* chg *n* EXT *n* ext *n*	MQCHG, *mqchg*, MQEXT, *mqext*, MQCHGR, *mqchgR*, MQEXTR, *mqextR*	quantitative mandatory change or extension, respectively, for in the future and in the past, respectively, for either entity type or relationships
EXT *n*	PCHG, PEXT, PCHGR, PEXTR	example of adding persistence to the transition; it is likewise for the rest

Figure 16: TREND Appendix B