The DISME low-code platform - from simple diagram creation to system execution

Vitor Freitas [1], Duarte Pinto [1], Valentim Caires [1], Leonardo Tadeu [1] and David Aveiro [1,2,3]

[1] *ARDITI - Regional Agency for the Development of Research, Technology and Innovation, 9020-105 Funchal, Portugal*
[2] *NOVA-LINCS, Universidade NOVA de Lisboa, Campus da Caparica, 2829-516 Caparica, Portugal*
[3] *Faculty of Exact Sciences and Engineering, University of Madeira, Caminho da Penteada 9020-105 Funchal, Portugal*

Abstract

An overview of the Dynamic Information System Modeller and Executor, an enterprise engineering, DEMO based, open source, low code platform with an Adaptive Object Model approach for workflow management in organizations. An alternative take, seeing an organization as a living organism, thus providing the tools for instant change in the processes to immediately reflect on their execution. We detail the multiple components of the System Modeller and how they interconnect with each other to produce the system executor that can be used by users for their day-to-day workflow in their organizations.

Keywords

low code platform, enterprise engineering, DEMO

1. Introduction

The Dynamic Information System Modeller and Executor (DISME) is a low code open source software platform that has DEMO methodology [1] as its foundation. It supports the production of collaborative-based organizational models and diagrams for the specification of its processes, information flow, responsibilities of both human and software, rules and other kinds of organizational artifacts.

These models and diagrams provide an up to date "picture" of the "organizational self" at any time and collaboratively, guiding its participants in: (1) supporting the perception of the global reality of the organization; (2) supporting the definition and execution of their operational work and; (3) supporting the creative process for organizational change.

DISME platform diverges from the usual low code approach because, instead of generating code for a static version of the organizational processes, it treats the organization as a living system, basing itself on the Adaptive Object Model [2], [3]. Models can be created, parameterized and can be made live immediately.

To achieve this, DISME has two main functional interfaces: (1) the System Modeler, that deals with the specification of the system through different diagrams, forms and tables; and (2) the System Executor that handles daily execution of processes and information flow, according to the specifications done in the System Modeller and the current state of the live information system of the enterprise running it.

The System Modeler in turn, can be divided into five main components:

1. Diagram Editor – where users can create (and/or view) a visual representation of the organization using DEMO models and then use those models to generate a great deal of the necessary content needed for generation of the organizational flow,

[1] CIAO! Doctoral Consortium, EEWC Forum 2022, November 2022, Leusden, The Netherlands
EMAIL: vitor.freitas@arditi.pt (V. Freitas); duarte.nuno@arditi.pt (D. Pinto); valentim.caires@arditi.pt (V. Caires); leonardo.tadeu@arditi.pt (L. Tadeu) daveiro@uma.pt (D. Aveiro)
ORCID: 0009-0002-0667-5749 (V. Freitas); 0000-0002-8451-5727 (D. Pinto); 0000-0002-0871-7212 (V. Caires); 0009-0005-0424-6301 (L. Tadeu) 0000-0001-6453-3648 (D. Aveiro)

© 2022 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-3388/paper4.pdf

Copyright © 2022 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2. System Specifier – the users are able to manage and shape each process of the organization by creating or editing transactions, their relationships and conditions and their associated documental support (like a form or need for a documento upload). Both the transactions and relations can be (almost) in their entirety inferred from the models diagram editor in correctly modeled, leaving the users to just fine tune their associated facts.
3. Action Rules Management – where users can specify how the system handles events that the actors have to respond to (business rules) and the work instructions regarding the execution of production acts.
4. Forms management – where users can visually create the templates for what the actors need to fill in any step of a transaction and then associate those forms to those transaction steps.
5. Dynamic Query management – where users can obtain query results in dynamic and user-friendly graphical interface using filters and basic operators to support satisfying their information needs.

The System Executor has just one main component, the Dashboard, where the users can navigate what is currently defined for each process according to their roles and permissions as well as be used to assign said roles and permissions and delegate tasks on other users.

To support all of this, DISME has a database heavily influenced by the DEMO way of thinking trying to capture and map what is the essence of an organization workflow but differing from the traditional approach as it does not abstract from the infological and datalogical aspects. While the database in itself is not tied to any specific way of working or tooling, the system still tries to offer the needed support by being open source and already integrated with free tools while also facilitating the integration to allow organizations to adapt it to their needs.

The database also follows the Type Square pattern [2] in order to allow for a clear separation of concerns regarding the system modeling and its execution.

Implementation wise, the latest iteration of DISME uses PHP (Laravel Framework[2]) on the backend, and in the front-end Typescript (Angular) to provide immediate feedback on any change by its users.

In the following sections we share an overview of DISME and the aforementioned components as well as current and future work.

2. System Modeller

The System Modeller is the “back end” of DISME where all the system specification and parameterization can be done by someone with knowledge of the organizational processes.

As mentioned before, it can be divided into five different components (seven if we account for smart contracts component and the Pages/Apps component still in their early stages) that together allow for the complete parameterization for the runtime environment as well as for increasing awareness of the organization (being it though the models, visual action rules or reporting)

We now explain each of the components that constitute the System Modeller at each subsection.

2.1. Diagram Editor

The diagram editor of DISME was inspired in the UEAOM [4] and uses drawio [3] (an open-source client-side editor for general diagramming) as a starting point.

This tool serves the purpose of presenting the visual representation of the implemented diagrams in DISME while being a fully functional editor and also to facilitate the design of new processes in a visual fashion converting (most of) them in an automated way into the database

[2] Laravel Framework - https://laravel.com
[3] Draw.io - https://github.com/jgraph/drawio

correspondents, instead of doing it in the System Specifier, by creating transactions and establishing connections.

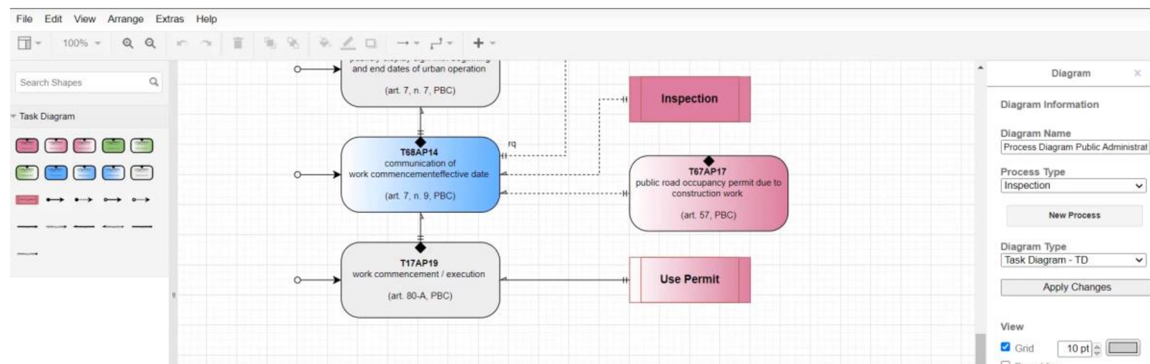

Figure 1: Task Diagram example in DISME's Diagram Editor.

In the Diagram Editor tool users have access to the traditional DEMO way of working diagrams as well as the proposed alternative notations for the process model in Figure 1 presented in [5] and the Fact Model in Figure 2 presented in [6]. Once the models are created users are able to convert their diagrams into the corresponding entries in the System Model in DISME's database. It is also possible, after designing (or editing) a process using the System Specifier, that users can also see their visual representation (automatically generated) in the diagram editor.

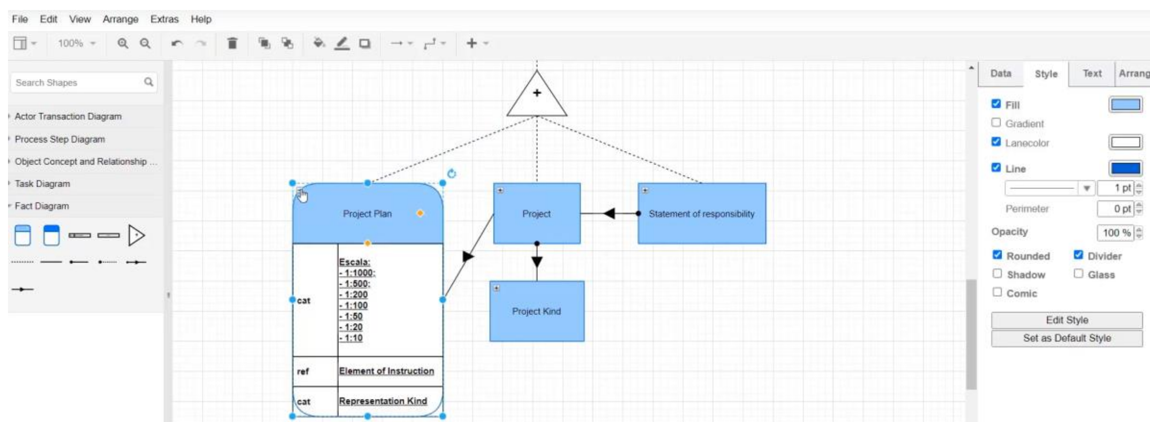

Figure 2: Fact Diagram example in DISME's Diagram Editor.

2.2. System Specifier

In the System Specifier, users are able to create/edit, in a table/forms based way, all model elements, having no need for any specific programming skill. They need only some basic knowledge of enterprise engineering modelling which is close to the "language / representation" used within organizations.

The System Specifier has the following sub-components:

Users Management – where new users can be created as well as current users' data can be changed, or users can be associated with roles.

Roles Management – where new organizational roles can be created or edited as well as assigned to users and to transactions. An organizational role can be associated with one transaction as executor and to several as an initiator.

Process Management – where processes are defined and modified. These processes are instantiated as runtime processes that a particular organization covers in its business area. The purpose in this component is to define the structure that the System Executor will then use to instantiate the flow of transactions that users participate in to accomplish their organizational goals.

Transaction Management – in this component, one specifies the transactions as traditionally captured in a DEMO approach. A transaction type needs to be associated with a process type, where instances of that transaction will be run by the system executor as part of the respective process instances.

Entity Management – where entity types (corresponding to DEMO's fact model classes) are created and modified. By specifying a set of entity types one is basically defining the main business concepts or, in other words, the concrete enterprise database tables.

Property Management – where the property types are defined and associated with the corresponding entity type they belong to. These property types, amongst other things, have a value type associated with them (text, int, enum, etc.).

Allowed Value Management – when the value type of a property is enum, it is necessary to also manage it by creating the possible values for the enumeration.

Units Management – units are another of the options that may be associated with a property. Their management (creation and modification) is done at a different place because the same units and their symbols (like for currency ($, €, …) or weight (kg, pounds, …)) will often be transversal to all properties that use them.

Causal Links – where the causal rules (as in DEMO) between transactions can be set. In other words, the connection between transactional acts that automatically originate the start (request transaction act) of another transaction. With these rules, the users at the system execution don't need to manually start a subsequent transaction in the process, they are automatically started following the defined process flow.

Waiting Links – where the causal rules (as in DEMO) between transactions can be set. These are set for specific transaction acts that are only allowed to continue in the flow of their transaction depending on the result of another specific transaction act.

It is to be noted that most of the specifications above will be possible to be made (some automatically) on the Diagram Editor, saving users much precious time. Nevertheless, having a simple form based interface to specify these elements is handy in different circumstances. The components above allow one to map most of the organization system elements but not in their entirety as there are no business rules applied to them nor the interface with the users. For these more intricate and complex specifications, we developed the components presented next.

2.3. Action Rules Management

The Action Rules Management component in DISME is used to create specifications of process logic, as well as different kinds of inputs and outputs with or without live user intervention. They follow the general principle of the Action Model of DEMO [7] .

The general proposed way in DEMO to represent an action rule is <event part> <assess part> <response part>. The event part specifies what event (or set of concurrent events) is responded to. The assess part in an action rule is divided in three sections, corresponding with the three validity claims: the claim to justice, the claim to sincerity, and the claim to truth. And the final part, the response, is divided in an if clause that specifies what action has to be taken if the actor considers complying with the event to be justifiable, and possibly what action must be taken if this is not the case. This way of formulating action rules allows the performer to deviate from the 'rule', if he/ she thinks this is justifiable (and for which he/she will be held accountable).

We considered this way of Action Rules Specification to be somewhat ambiguous because, although it uses a structured English syntax similar to the one used in Semantics of Business Vocabulary and Rules [8] it does so in an incomplete way that does not contain all the needed ontological information to derive the implementation from it. For example, it lacks a way to deal with sets of actions or operators. This set of rules was also complex by containing mostly unneeded specifications of three types of assessment, the justice, sincerity and truth. For these reasons, the approach taken to the Action Rules component in DISME was the one proposed in [9].

From an implementation standpoint, after analyzing the open-source options that could be used to specify Action Rules in a user-friendly way, the Blockly library was chosen.

Blockly provides a visual code editor to web and mobile apps. The Blockly editor uses interlocking, graphical blocks to represent code concepts like variables, logical expressions, loops, and more. It allows users to apply programming principles without having to worry about syntax or the intimidation of a blinking cursor on the command line.

The goal is for the end-user to specify the action rules for the business case at hand without needing to have programming knowledge and in a visual manner. An example of a business rule built with Blockly can be seen in Figure 3.

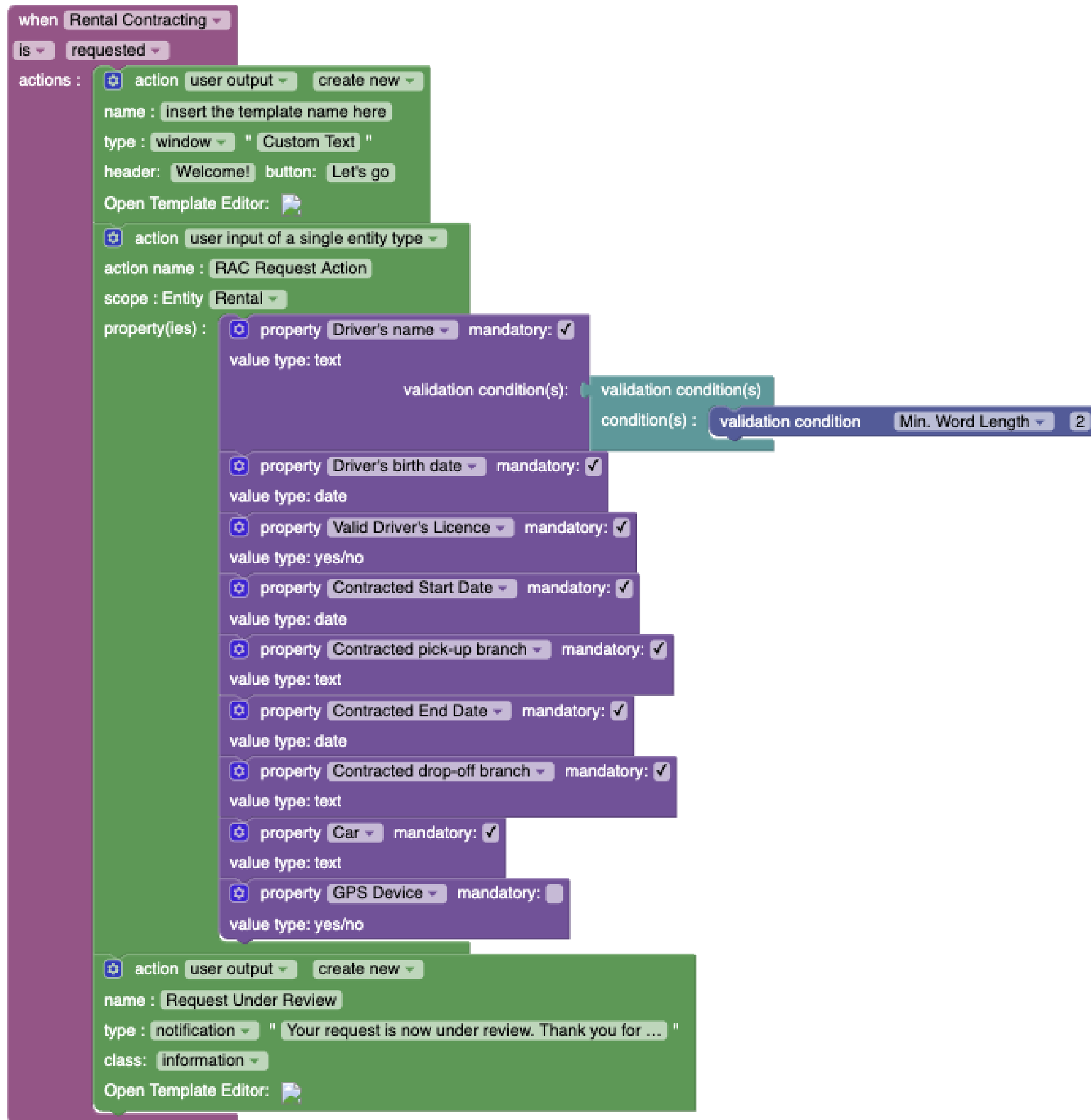

Figure 3: Blockly action rule to deal with the request of a rental contract.

The specified action rules are parsed and stored in DISME's database and are dynamically interpreted by DISME's execution engine according to the initiatives of users and/or external systems. Each action rule can have one or more actions. These actions occur in the context of a transaction type at the activation of a particular transaction state. Each transaction type and transaction state can perform multiple actions, but these actions belong to a single action rule.

Actions must be of a certain type which will determine the expected behavior of the system. The execution of these actions is always in sequence, according to the current flow of an action rule. The currently allowed action types for are the following;

- **user_input** - An action type of user input means that a user must fill some information inside a form. This action type is always executed through the presentation of a form with fields where a user must fill or choose (depending on the field type) information.
- **user_output** - An output information presented in the dashboard to a user/client through notifications or popups.
- **assign_expression** - With this action the engine can automatically save/write a value to a specific property of a specific entity.
- **causal_link** - The main function of this action is to generate a new transaction instance or to change the t_state of the current transaction.
- **if** - This type is used for specifying one or more statements to execute depending on the result of a condition evaluation. After this evaluation, it should perform one or more actions depending on the result. For example, with this type, we can control the action flow of a transaction instance.
- **then** - This action is used to aggregate multiple actions when the if expression evaluates to true.
- **else** - This action is used to aggregate multiple actions when the if expression evaluates to false.
- **PRODUCE_DOCUMENT** - This action generates a document based on a template previously created by the user. This template for the document can be dynamic, meaning that it can have variables inside that are properties from a form. **EXTERNAL_CALL** - the objective of this action is to interact with external systems using some API.

This last set of Actions make use of most mentioned tables of DISME's database to produce the action rules that allow for the system to be executed as intended following the business logic. All this as mentioned before can be created visually with the help of Blockly with little more than some knowledge of logic expressions by dragging and dropping the pieces into the canvas (Blockly options can be seen in Figure 4 and combining them together. The action rules management component validates both syntax and semantics while being used, so the user is guided through the whole process.

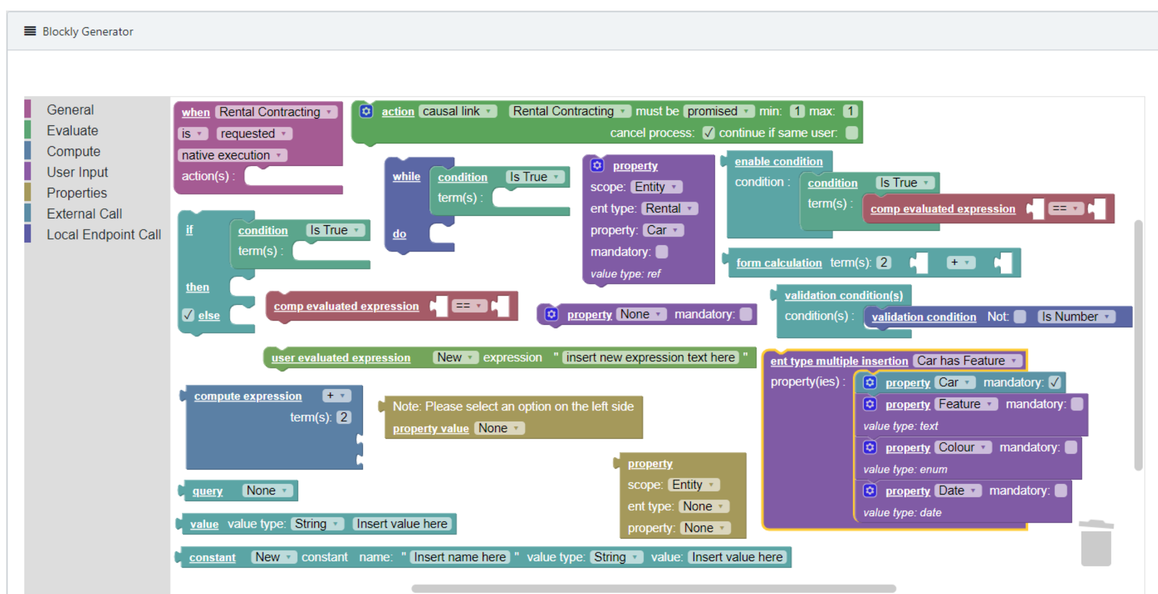

Figure 4: Blockly pieces that can be used for the creation of Action Rules.

After the specification of all internal process logic and structure of inputs/outputs one needs to deal with the interface. In DISME this is done with the Forms management component, described next.

2.4. Forms Management

In order to develop the Form Editor component, research was made about several plugins that enabled a form editor to be integrated in an Angular frontend, the one currently being used in DISME's architecture. The plugin ultimately chosen was Form.io[4], an open-source form editor that was completely integrated into DISME, with the customization of several functionalities to satisfy the concrete specifications and needs for its use in DISME.

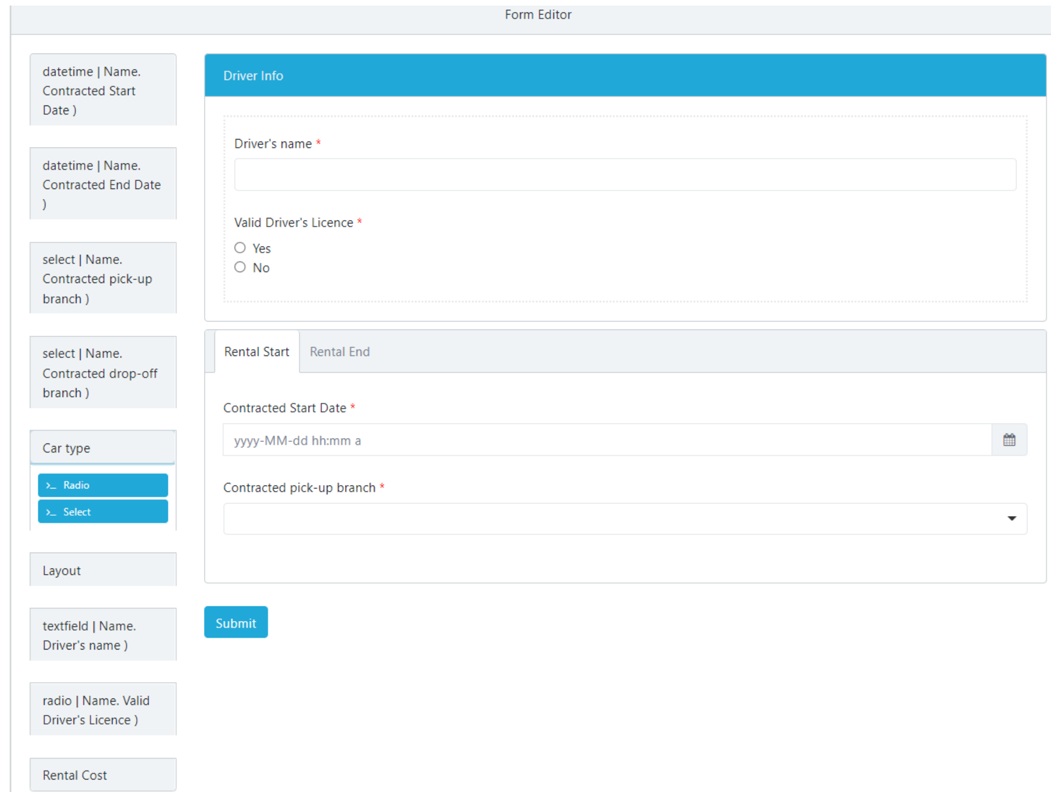

Figure 5: Form creation using the integrated Form.io.

The use of the Form tool is rather simple, the user selects the form field types they want to use and adds it to a canvas in a place of its choosing. Because the type of information being collected has already been specified in the Action Rules Management component, it is possible to already give the user the exact kind(s) of input that can be used, thus accelerating the Form building process, as seen in Figure 5. The layout of the form and how it should be presented to the final users is totally up to its creator to design. Once the Form creator is happy with the Form, he can just save it, and if the process is active, it automatically becomes available for execution.

In Figure 6 we have an example of the form that was being created in Figure 5 being executed in the dashboard.

At this point an organization can already run their processes in the DISME. But there are still more features that were developed to facilitate the use of DISME as a rich information system. The capability of dynamically designing queries is one of them, presented next.

[4] Form.io - https://form.io

Figure 6: Form to be filled on transaction step in the Dashboard.

2.5. Dynamic search

The dynamic search component works in the way of specification of queries based on triplets of property-operator-value, chosen by the user selecting the relevant options in a user-friendly graphical interface and without the need of any programming experience (e.g. SQL). This specification is done in 4 different steps. An explanation of each of these steps follows, together with screenshots.

Step 1. Select an Entity Type

Figure 7: Entity selection panel for Dynamic Queries.

In the first step of the query specification process, one has to select the entity types that the search will be focused on. The first selected entity type will serve as the "Base Table", that is, as the main entity type where we will be searching for entity values based on the filters defined ahead. After selecting the Base Table, the entity types that are referenced by it will be available for further selection. These entity types will be then treated as if they were "Related Tables". The

entity types that aren't related to the Base Table will have its checkbox disabled and can't be selected. An example of the selection available in this step can be seen in the Figure 7.

Step 2. Select Properties

After having established the entity types that will be included in the search, it is time to define the properties. At this step, for each entity type selected, there are 2 select boxes to define the Included Properties and the Filter Properties. For each one, the options shown in the select box are the properties belonging to that entity type. The Included Properties are the properties that will be shown in the results table. The selected properties will define the result table's rows. Important to mention, as well, that for each property selected in a Related Table's select box, there must be a selected referenced property in the Base Table's select box. For example, again in Figure 8, if we select the Rental Tariff per Day property in 'Car Type', we must select a property from 'Rental' that references it. In this case, that means selecting the Car Type property. The same can be applied to selecting the Location property in 'Branch'. In this case, as there are several properties in 'Rental' that reference Branch, we must select which of these we want in the result. In this case, the Contracted Pick-Up Branch was selected. Multiple referenced properties can also be selected instead of just one.

The Filter Properties are the properties that will be used in the restriction of the results of queries based on triplets of property-operator-value. The properties selected in these select boxes will then be available for specifying filters in Step 3. In case there are no properties selected, the user can still choose to see the results' table, bearing in mind that there will be no filters applied to any properties and consequently the results will include every instance of the Base Table's entity type present in the database.

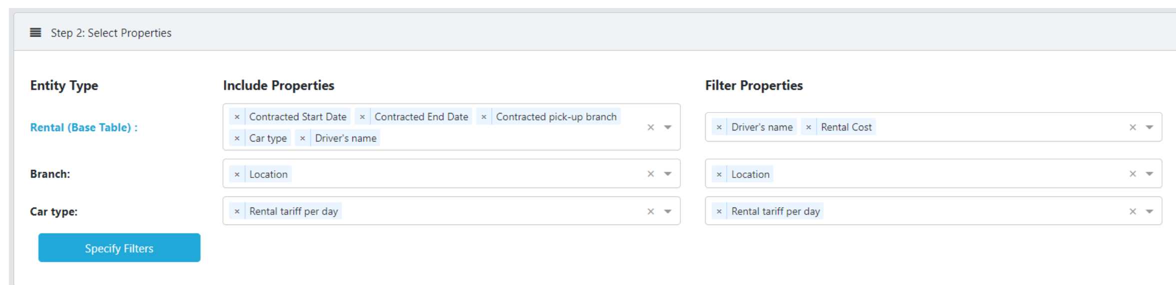

Figure 8: Property selection panel for Dynamic Queries.

Step 3. Specify Filters

After having selected the Filter Properties to consider, in this step one defines the concrete filters to apply. This functionality was implemented taking advantage of the Angular Query Builder component.

Here, one can define rules and rulesets that will be applied to the main query to be run in the database. It can be seen as rules being conditions and rulesets being sub-conditions. Whether one or another is selected, a type has to be chosen - AND/OR. One must also define the property to be filtered, the query's operator and the value that will restrict the result.

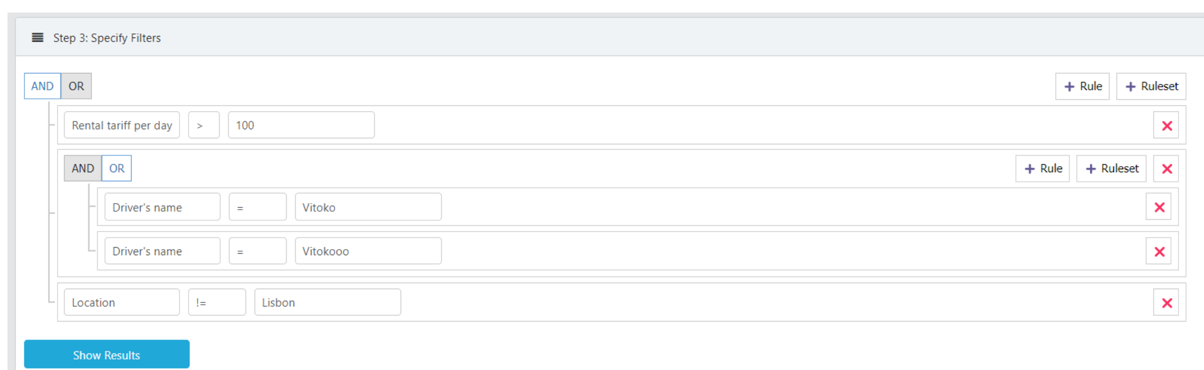

Figure 9: Filter application panel for Dynamic Queries.

An example of its usage is shown in the Figure 9, where we have a parent query of type AND that has 2 rules (conditions), and one ruleset (sub-condition). That ruleset defined is of type OR. The query defined in the image can be translated to:

[Rental Tariff per day > 100] AND
[Driver's name = Vitoko OR Driver's name = Vitokooo] AND
[Location != Lisbon]

Step 4. Query Results

After having Steps 1, 2 and 3 completed, the user can choose to see the query results through the button displayed in Step 3's image. When it is pressed, a table with the query results will be shown, as can be seen in Figure 10, that refers to the results of the query applied in Step 3.

Note that the Included Properties selected in Step 2 are what defines the table's rows. For Base Table's properties, we have the table header as the name of the selected property. For Related Table's properties, we have the table header as the junction of the selected property and the name of the Base Table that it is referencing.

Step 4: Query Results

Export Results to Excel

Contracted Start Date	Contracted End Date	Contracted pick-up branch	Car type	Driver's name	Contracted pick-up branch: Location	Car type: Rental tariff per day
2022-01-08T12:00:00+00:00	2022-01-22T12:00:00+00:00	Berlin Airport	Intermediate	Vitoko	Berlin	200
2022-01-22T12:00:00+00:00	2022-01-22T12:00:00+00:00	Berlin Airport	Luxuary	Vitoko	Berlin	350
2022-01-27T12:00:00+00:00	2022-01-21T12:00:00+00:00	Berlin Airport	Intermediate	Vitoko	Berlin	200
2022-01-08T12:00:00+00:00	2022-01-28T12:00:00+00:00	Paris Airport	Intermediate	Vitoko	Paris	200
2022-01-06T12:00:00+00:00	2022-01-29T12:00:00+00:00	Funchal Airport	Standard	Vitokooo	Funchal	250

Figure 10: Dynamic query result.

Contracted Start Date	Contracted End Date	Contracted pick-up branch	Car type	Driver's name	Contracted pick-up branch: Location	Car type: Rental tariff per day
2022-01-08T12:00:00+00:00	2022-01-22T12:00:00+00:00	Berlin Airport	Intermediate	Vitoko	Berlin	200
2022-01-22T12:00:00+00:00	2022-01-22T12:00:00+00:00	Berlin Airport	Luxuary	Vitoko	Berlin	350
2022-01-27T12:00:00+00:00	2022-01-21T12:00:00+00:00	Berlin Airport	Intermediate	Vitoko	Berlin	200
2022-01-08T12:00:00+00:00	2022-01-28T12:00:00+00:00	Paris Airport	Intermediate	Vitoko	Paris	200
2022-01-06T12:00:00+00:00	2022-01-29T12:00:00+00:00	Funchal Airport	Standard	Vitokooo	Funchal	250

Figure 11: Exported excel file from a Dynamic Query.

Furthermore, it's also possible to export the results obtained in this step to an Excel File, through the button that can be seen in the top right corner of the image above. An example of the resulting file can be seen in Figure 11.

3. System Executor

All users, when logged in DISME are directed to the Dashboard where a list of the tasks they are allowed to perform is shown.

A user can execute a request to start some specific process or react to a certain process state to which they were given authority and responsibility to do so – if some property/entity is associated with that act the user will have to fill out a form, automatically generated based on the previously specified parameters on the System Modeller.

The Dashboard allows the users to control the flow and state of all process instances and data, thanks to both the causal and waiting links that were specified in the respective modeling functions and the data submitted by the users. The other main component of the System Executer is the Execution Engine which is triggered by interactions on the Dashboard or certain foreseen events in the system specification. For space reasons, in this paper we focus mostly on the Dashboard.

3.1. Dashboard

DISME dashboard was inspired by traditional BPM platforms that share the common goal, such as: Camunda [5] , Bonita [6] , Process Maker [7], among others.

This dashboard allows the initialization of processes, execution of tasks dependent on an existing process, and allows for better management of pending, completed and delegated tasks, as well as the continuation of pending tasks, whether they have been delegated or not as long as under this user responsibility.

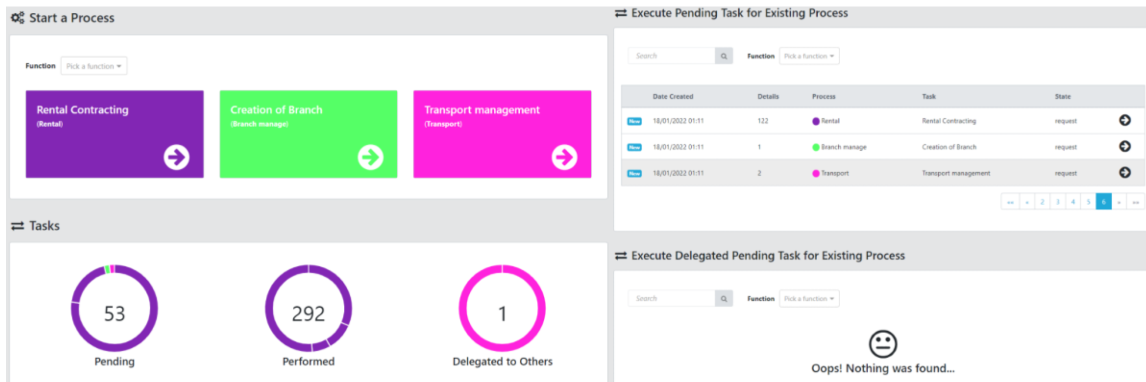

Figure 12: DISME Dashboard presentation for a user with processes, pending tasks, task statistics and delegations.

In Figure 12 we can see an example of a dashboard for a rent-a-car case on the management side.

In that menu we have a process control area for the process types that are defined. These process types make up the set of processes that a particular organization covers in its business area. The purpose of the process type is to define the structure that the system execution part of the platform uses to create the instances of processes that occur within a company.

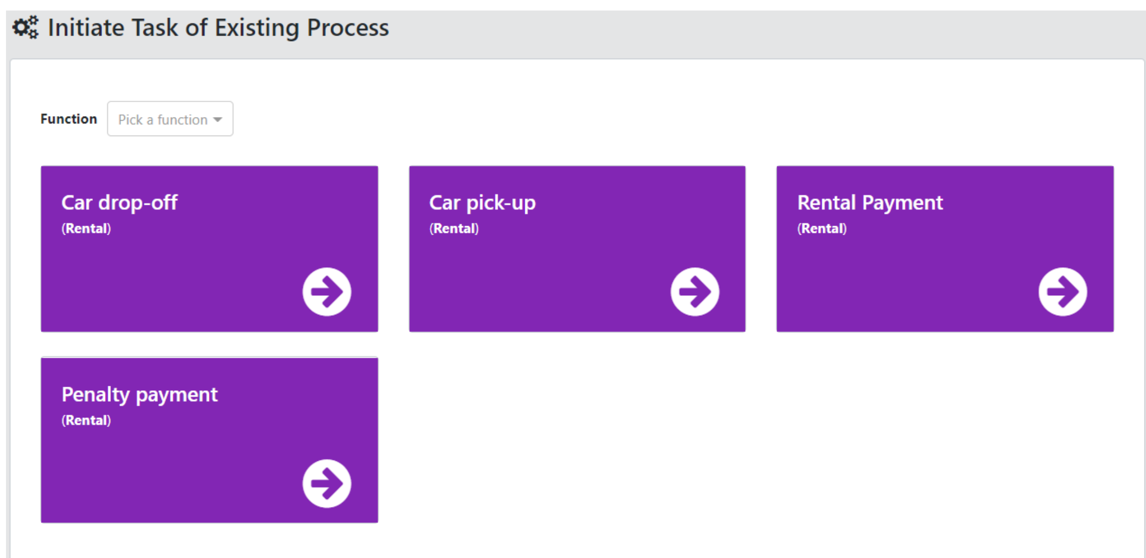

Figure 13: DISME Dashboard - list of initiated transactions for a user in the car rental process.

Each Process Type is nothing but a collection of related transaction types. The purpose of these process types is to define the structure that the system execution part of the platform uses to

[5] Camunda - https://camunda.com

[6] Bonita Platform - https://www.bonitasoft.com

[7] Process Maker - https://www.processmaker.com

create the instances of processes that occur within a company. An example of a process type is Rental that relates the following transaction types: Rental Contracting, Rental Payment, Car Pick-Up, Car Drop-Off and Penalty Payment.

In Figure 13 we have a dashboard example for that example with a user who is in the rental process (in this particular case with multiple options, for showing purposes, but in reality would only have the available ones clickable).

Because there is a clear separation between the backend and frontend, all the information used is gathered through the Laravel implemented rest API, so it is possible for any organization to use an alternative customized layout or even integrate DISME process management (the system execution logic) with their already existing tools through the use of a vast collection of API endpoints.

4. Conclusions, ongoing and future work

DISME intends to be a disrupting offering in terms of low-code platforms with some clear differentiators from most other options such as: the DEMO foundations or the Adaptive Object Model approach. With DISME organizations don't risk vendor lock in as everything it uses is open source as is DISME itself. While still in development, the platform already offers a high degree of working functionality. The first alfa version of the platform will be available on github soon[8].

Besides the previously presented components which are completed or nearing completion, there are other important functionalities being worked on.

One of those is the Electronic Document Management System component, based on Mayan EDMS[9] which offers features such as a REST API, optical character recognition, indexing and text searching, metadata of the document, digital signatures, etc. This will allow richer management of documental inputs and outputs in the platform.

Another feature being worked on is the *cockpit manager* to allow the specification and use of standalone *pages* or *apps*, including a mix and match of elements of other components, namely, HTML blocks, forms, results of dynamic queries, process initiation, etc.

We are also developing a Rapid REST API management component. It will allow the creation of endpoints made available to external systems, to provide simple lists of data items or even the results of complex queries or operations, as well as enacting internal tasks on the system based on calls from external systems. Furthermore, it will also automatically scan data provided by external APIs that our system can call and match with internal data,, facilitating the integration of external information into our local system. This will be done by simple drag and drop operations in a friendly GUI.

Yet another feature being worked on is a Smart Contract generation and blockchain integration component. This component will allow a high-level DEMO based way of specifying smart contracts and a method for automatically generating smart contracts in Chaincode, using Hyperledger Fabric as the blockchain platform. We are developing mapping from DEMO Action Rules language to Hyperledger Chaincode using GO. This will facilitate the adoption of smart contracts in business processes and contribute to enterprise interoperability supported by blockchain technology. Taking advantage of having Blockly, already integrated with DISME for the generation of Action Rules, we will be extending it to include the creation of Smart Contracts in the business logic as an extension to those rules. This approach will not only reduce the manual labor involved in the generation of the smart contracts but also reduce the likelihood of errors as the blocky component already does both semantic and syntax validations.

The near future work in the platform will be mostly in concluding the features being implemented, fine-tuning the ones already implemented and guaranteeing their full interoperability. The platform needs thorough testing which will be realized soon in the context of two practical research projects, where DISME will be used to create an information system for the respective organizations which are partners in these projects.

[8] https://github.com/orgs/EnterpriseEngineeringLab/projects

[9] Mayan EDMS - https://www.mayan-edms.com

5. References

[1] J. L. G. Dietz and H. B. F. Mulder, 'The DEMO Methodology', in *Enterprise Ontology: A Human-Centric Approach to Understanding the Essence of Organisation*, J. L. G. Dietz and H. B. F. Mulder, Eds., in The Enterprise Engineering Series. Cham: Springer International Publishing, 2020, pp. 261–299. doi: 10.1007/978-3-030-38854-6_12.

[2] J. W. Yoder, F. Balaguer, and R. Johnson, 'Architecture and design of adaptive object-models', *SIGPLAN Not*, vol. 36, no. 12, pp. 50–60, Dec. 2001, doi: 10.1145/583960.583966.

[3] J. W. Yoder and R. Johnson, 'The Adaptive Object-Model Architectural Style', in *Software Architecture: System Design, Development and Maintenance*, J. Bosch, M. Gentleman, C. Hofmeister, and J. Kuusela, Eds., in IFIP — The International Federation for Information Processing. Boston, MA: Springer US, 2002, pp. 3–27. doi: 10.1007/978-0-387-35607-5_1.

[4] D. Aveiro and D. Pinto, 'Universal Enterprise Adaptive Object Model: A Semantic Web-Based Implementation of Organizational Self-Awareness', *Intell. Syst. Account. Finance Manag.*, vol. 22, no. 1, pp. 3–28, 2015, doi: 10.1002/isaf.1363.

[5] D. Pinto, D. Aveiro, D. Pacheco, B. Gouveia, and D. Gouveia, 'Validation of DEMO's Conciseness Quality and Proposal of Improvements to the Process Model', in *Advances in Enterprise Engineering XIV*, D. Aveiro, G. Guizzardi, R. Pergl, and H. A. Proper, Eds., in Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2021, pp. 133–152. doi: 10.1007/978-3-030-74196-9_8.

[6] B. Gouveia, D. Aveiro, D. Pacheco, D. Pinto, and D. Gouveia, 'Fact Model in DEMO - Urban Law Case and Proposal of Representation Improvements', in *Advances in Enterprise Engineering XIV*, D. Aveiro, G. Guizzardi, R. Pergl, and H. A. Proper, Eds., in Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2021, pp. 173–190. doi: 10.1007/978-3-030-74196-9_10.

[7] 'DEMO Specification Language 4.6.1 – Enterprise Engineering Institute'. https://ee-institute.org/download/demo-specification-language-4-6-1/ (accessed Mar. 28, 2023).

[8] P. Bollen, 'SBVR: A Fact-Oriented OMG Standard', in *On the Move to Meaningful Internet Systems: OTM 2008 Workshops*, R. Meersman, Z. Tari, and P. Herrero, Eds., in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 718–727.

[9] D. Aveiro and V. Freitas, 'A new Action Meta-Model and Grammar for a DEMO based low-code platform rules processing engine', presented at the EEWC 2022, Leusden, Netherlands, Leusden, Netherlands, 2022. doi: Forthcoming.