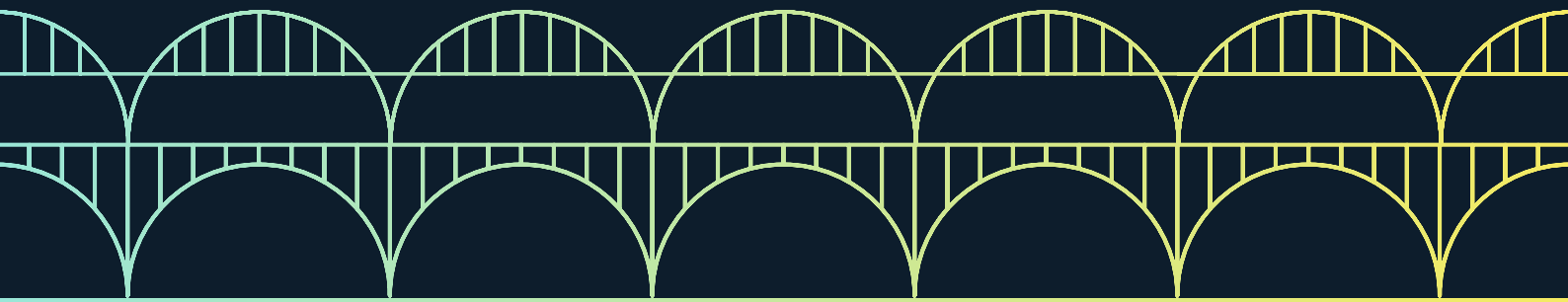
MDENET RESOURCES

MDENet | The home of model-driven engineering

MDE for Tax & Payroll Calculations



Federico Tomassetti, Steffen Zschaler and Meinte Boersma

The MDENet "MDE for ..." series introduces ways in which MDE can be applied to address challenges in a wide range of different domains. Intended to help you identify how MDE might help you, each report identifies some key challenges in the domain, describes potential MDE applications through concrete case studies, and clarifies some of the key MDE concepts that will help address these challenges.

In this document, we want to collect some examples of important organizations which have used MDE to change the way in which they developed software for tax and payroll calculations. This approach is based on Domain Specific Languages (DSLs). DSLs permit better collaborations between experts in tax system and payroll calculation and developers, and using them results in faster development cycles, reduced number of errors, and reduced development costs.

We will see four different cases:

- Sistemi, the market leader in Italy for tax consultants software
- A large multinational, present in over 50 countries (name is confidential)
- The Dutch Tax and Customs Administration (Belastingdienst)
- DATEV, a German entity with over 400,000 customers

A leading company in Italy for tax consultants software

We have worked with Sistemi, a leading Italian company in the software for tax consultants market.

This company has over 40 years of history. They started developing software for tax consultants a very long time ago. Decades ago they made a choice that has been paying off: they developed a proprietary language to define specific concepts related to their domain. So their language had support for concepts like quarters and tax brackets, which are not typically part of a common programming language.

The Challenges

Challenge 1: Supporting multiple platform

From the early stage, the company needed to support multiple platforms. While the platforms changed over time the need to support more than one remained constant. With traditional development techniques that would have required a significant increase in development costs and possibly a reduction in quality, as each implementation would have been tested by fewer users.

Challenge 2: Technological transformation

The technologies used to deliver the application logic varied in time. For example, the applications transformed from desktop applications into web applications. With traditional development techniques that would have required a complete rewrite of the applications, at a very high cost and an unsustainable time to market.

Challenge 3: Ensuring continuity

Compatibility with the past was a strong requirement. This was very difficult to achieve when paired with the evolution of technological platforms. With traditional software development techniques this would have brought unsustainable costs.

Why is this important?

Because code written in their proprietary language survived across several technological changes. The competitors that shared such a long history and they did have to regularly rewrite their software over time.

This company instead originally wrote applications that had a terminal interface - do you remember the 80x25 screens? Over time they needed to replace the software with desktop GUIs for Windows machines. And then 10-15 years later the time came to move to web applications.

And they did, but without the need to rewrite the logic for tax calculations. How is that possible? They simply worked on their DSL engine and adapted it to produce desktop applications first and web applications later.

This meant that they did not have to stop, and redo their entire core product, with a huge investment and a significant slowdown in development.

In their case adopting a DSL for developing the solution meant the knowledge poured by tax and payroll experts was still reusable even when technologies changed. This is a major difference with traditional software development, where software has to be rewritten to accommodate major technological transformations.

The Benefits

Decoupling logic and technology implementation

Separating logic from technology implementation enabled changing technology while maintaining logic: a routine that was written 40 years ago is still valuable even though the underlying tech stack has changed completely.

Supporting multiple platforms

By combining a DSL with multiple generators, it was possible to support multiple targets without increasing costs.

A large multinational, present in over 50 countries

A few years ago we worked with a large multinational that was producing a wide range of software for tax consultants, accountants, and payroll processing. This company was and is still present in over 50 countries. And some of those countries presented different rules for different areas: think about the federal states like the USA or Germany, with different taxes for each state. Or countries like France, with special rules for "Metropolitan France" and their territories far away from Europe.

These companies have to face very significant challenges due to their business. On the basis of a single platform, they need to have specialists from all over the world define the specific rules for each jurisdiction, and keep them up-to-date year after year. Tackling these challenges with traditional development techniques can prove very expensive.

The Challenges

Challenge 1: Handling variability

The challenge faced was in how to reuse one technology stack in different legal environments, each of which has their own set of tax rules which need to be captured and implemented faithfully.

Challenge 2: Supporting different teams of domain experts

A centralized development team had to support multiple domain experts teams, working in different natural languages.

The MDE Approach

This company therefore developed its own language, which specialists can use autonomously to define the logic for tax calculation and verify it. Different experts all over the world can get easily accustomed to it in a matter of hours.

This reduced the costs for making experts and developers communicate, sped up development, and reduced a lot of errors as less misunderstandings are possible.

In this case a DSL permitted to have a scalable approach to capture knowledge about tax calculations and have this knowledge integrated with the software platform developed by the technical team.

The Benefits

Agile evolution of business logic

Separating logic and technology allows flexibly exchanging the logic on top of the same technology stack

Validation by domain experts

Capturing the logic using an explicit modeling language (a DSML) these can be captured more easily and validated by domain experts in the relevant jurisdictions.

Parallelization of development

By adopting a DSL, domain experts were able to work independently. This had the potential for reducing bottlenecks because the central technology team does not have to develop each system for each new jurisdiction.

The Dutch Tax and Customs Administration (Belastingdienst)

The Dutch Tax and Customs Administration has to face one important challenge: to update their software every year, based on the changes in fiscal legislation.

One has therefore to study the tax code (Belastingwetten), examine the difference with the support of tax and legal experts, and consequently update the software in a short time.

If we consider that hundreds of billions of Euros are at stake, as the tax returns of millions of citizens are processed, we can understand the importance of getting calculations exactly right.

The Challenges

Challenge 1: Complexity and frequent changes

The tax code changes every year, and changes are complex to interpret. They can also have consequences which are not easy to foresee.

Challenge 2: Time constraints

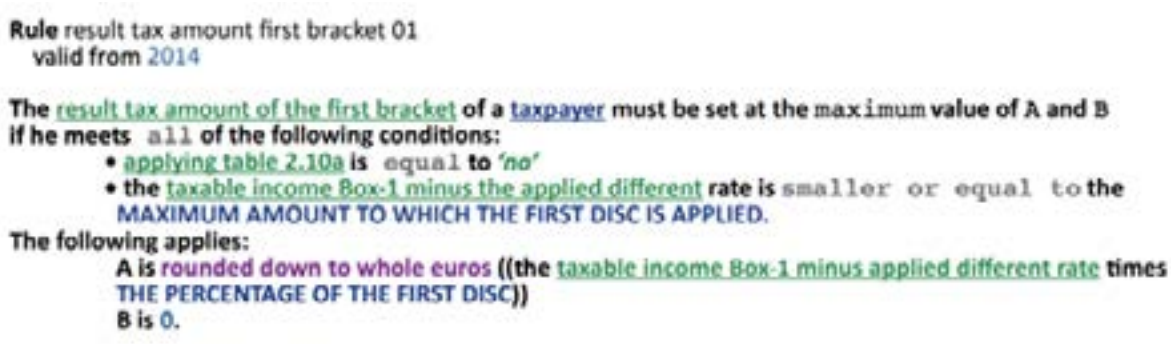
Every year tax returns have to be calculated according to a pre-determined schedule, therefore the time available to perform changes to the code is limited and cannot be extended.

The MDE Approach

The Dutch Tax Agency has defined a standard for controlled Dutch called RegelSpraak. With RegelSpraak, rules for tax calculations (extracted from the tax law / code with the intent to automate) can be defined in a precise and unambiguous way that lends itself to direct automation. Later on, the Dutch Tax Agency developed ALEF, an IDE for several DSLs, including an implementation of RegelSpraak, that allow rule analysts to specify complete taxation systems that require zero coding to deploy to production.

Citing Diederik Dulfer, Architect Business Rules Management at Dutch Tax and Customs Administration, “MDE is the future and the Tax and Customs Administration is already well on its way!”.

The Dutch Tax and Customs Administration (Belastingdienst)



```
Rule result tax amount first bracket 01
  valid from 2014

The result tax amount of the first bracket of a taxpayer must be set at the maximum value of A and B
if he meets all of the following conditions:
        • applying table 2.10a is equal to 'no'
        • the taxable income Box-1 minus the applied different rate is smaller or equal to the
          MAXIMUM AMOUNT TO WHICH THE FIRST DISC IS APPLIED.
The following applies:
        A is rounded down to whole euros ((the taxable income Box-1 minus applied different rate times
        THE PERCENTAGE OF THE FIRST DISC))
        B is 0.
```

In the case of the Dutch Tax Administration, because of the DSL it is very easy to trace the tax rules written by legal experts to the original laws, and when laws are modified, changes in the DSL code can be applied with limited effort and with confidence. Tax rules act as their own historical record, because they are explicitly versioned with each version having a validity range. In addition, tests can be written directly next to the rules under test, and these tests are executed by the ALEF tool all the time.

The Benefits

Make Domain Experts independent

Providing a set of DSLs that closely mimic how the tax code is written enables civil servants to write, deploy and update (core parts of) taxation software systems independently.

Reduce costs

Using ALEF reduces the communication necessary between civil servants and software developers significantly, reducing costs across the whole process. (This includes opportunity costs due to developers being hard to find.)

Built-in validation facilities

Enabling civil servants to validate their work through explicit tests which are continuously running. That enables civil servants to check that their implementation matches the (intention of the) tax law.

Explainability

Using tax rules written with ALEF means that explaining tax calculations can be automated as well: citizens can be given a detailed, and thorough explanation of how their taxation is built up, and why rules were applied and how.

DATEV, a German cooperative with over 400,000 customers

DATEV eG has more than 400.000 customers for its service on tax and payroll slips calculations. Founded in 1966, the organization now has about 8.100 employees, working in its headquarters in Nuremberg and 23 subsidiaries in Germany.

Over 13 million payroll slips are processed every month by DATEV.

DATEV has interesting challenges, like for example supporting the ability to recalculate payslips considering the laws and regulations that were valid at a particular point in time in the past.

The Challenges

Challenge 1: Temporal variability

Most data in this domain varies over time. For example, salaries or tax rates. Most operators (e.g., +, -, *) must therefore take this into account, which significantly adds to the complexity of the implementation.

Challenge 2: Tight implementation deadlines

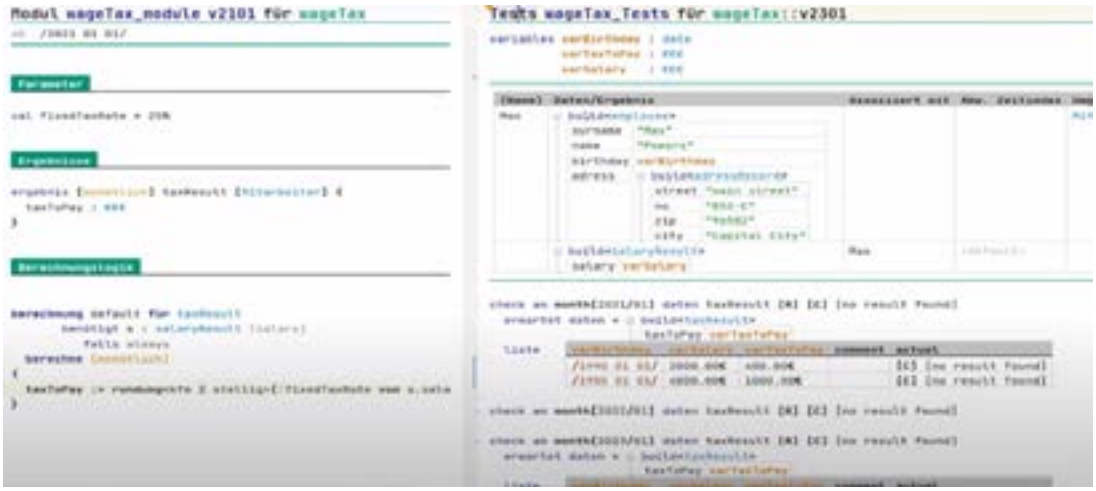
Changes to the rules are outside the control of DATEV but may need to be implemented correctly in a short timeframe (weeks) to then be applied to over 13 million payroll slips.

The MDE Approach

The most important advantage of this approach is that it permitted to better separate concerns: developers could focus on technical aspects, like the cloud infrastructure, while experts were able to define rules for tax and payroll calculations autonomously.

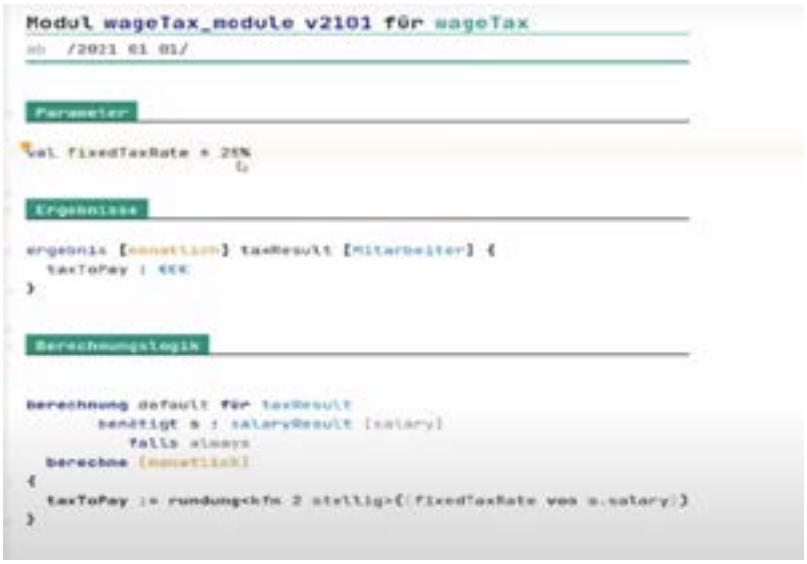
Most calculations have also to take into account not only regulations but also data changes, like for example salaries being increased during the year. Temporal dimensions are difficult to track in traditional software development but are addressed much more easily in the DSLs developed at DATEV.

It should also be noted that the DSL can use terms in the natural language which is most convenient for the users. For example German or Italian. Typical programming languages are instead restricted to English keywords.



This images comes from [4]

DATEV, a German cooperative with over 400,000 customers



This images comes from [4]

The Benefits

Domain logic and technology separation

Separating logic and technology brings the usual benefits as discussed in the other case studies.

Localized support

The DSL can use natural-language terms that are easier to use for domain experts than the typical English terms used in programming languages.

Explicit support for temporal variability

By allowing expressions in the DSL to be tied to particular timeframes, there was a powerful framework for capturing the complexity in time of calculating payroll slips.

2. KEY CHALLENGES

What key challenges are there in Tax and Payroll that could benefit from MDE?

MDE and DSLs could help tackle some of the main challenges faced when implementing tax and payroll calculations with traditional development techniques.

While there are several challenges we consider useful to underline these three:

1. Handling variability

Variability is very high both because the regulations change continuously but also because they vary in different jurisdictions.

2. Time constraints

Typically changes in regulations need to be reflected into software in a timely manner. This was for example the case during the pandemic, with different governments and regulatory agencies enacting changes in regulations with immediate effect.

3. Coordination between developers and domain experts

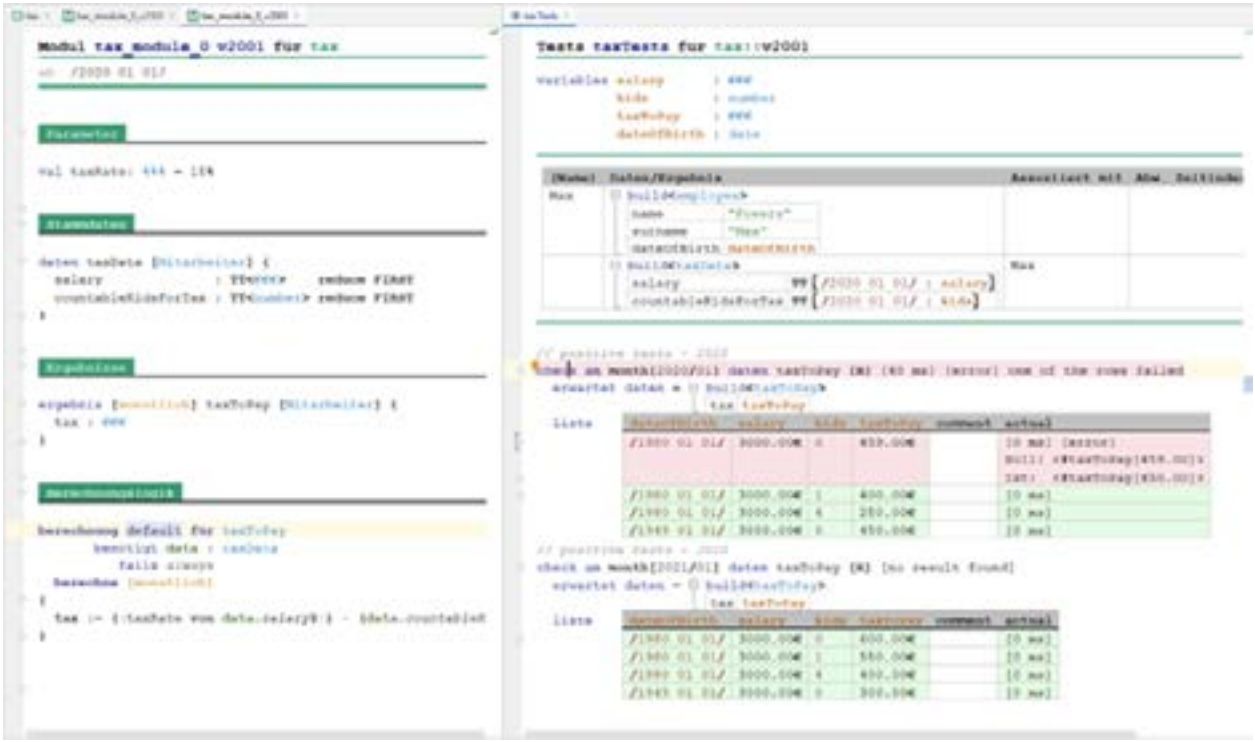
It is costly to coordinate developers with domain experts. They need to learn how to communicate with the other party without sharing a common background. They are often prevented to work independently because each rely on the other in order to progress with the process of reflecting in software tax and payroll regulations.

3. HOW CAN MDE HELP?

How can MDE concepts help address these challenges?

A Domain Specific Language for tax and payroll calculations permits experts to capture their domain consisting of tax brackets, rules for deductions, withholding regulations, and so on.

Think of a specialized version of Excel, which understands the domain and can guide the user, avoiding a lot of possible mistakes and simplifying the work.



This images comes from [3]

In traditional software development, experts have to explain to concepts analysts, analysts write them down in large functional requirements documents, which are then implemented by software developers. This whole approach requires a lot of coordination and causes issues, as the developers are not accountants, and the accountants are not developers. Feedback cycles are long and misunderstandings typically emerge only after developers have completed their work.

DSLs change all of that, making it possible for experts and developers to collaborate much more efficiently, with everyone working on what they can do best.

Where are we? Problems solved and challenges

The problem we know how to solve with DSLs is to make it possible for tax and payroll specialists to formalize their understanding of regulations and calculations without the intermediation of developers. We give them languages and supporting tools.

What advantages does this bring with respect to having expert writing requirements and developers formalizing them into code?

This means:
• Reduced development time, which in turn means lower cost and faster time to market.
• Reduced costs, as errors are found while writing specifications using the DSL while traditionally errors are found much later, during development and are therefore more costly to correct.
• Reduce communication costs, as we do not need meetings to have tax and payroll experts explain things to developers and vice versa. We also avoid all the misunderstandings that often happen when these two groups of people have to work together.

We know how to design these languages, and how to create the supporting tools, and we have experiences telling us they work in practice.

Let's start with the bad news: as of now there is no DSL for taxes and payroll that you can simply buy. You need to build one for yourself, or get it built for yourself by someone with experience on this. That would of course mean that you will end up with a tailored solution, specifically designed for your specific needs.

So, what resources can help you in this journey?

You may want to learn more about DSLs, for that we suggest reading The Complete Guide to (external) Domain Specific Languages.

After reading it, you should get a much better idea of what DSLs are, what benefits they bring, and also some ideas of the alternative solutions to build them.

If you want to dig deeper and learn how to actually design DSLs for yourself there are two books we would recommend:
Business Friendly DSLs will help go straight to the point and teach you how to build lightweight DSLs running right in the browser. No time wasted in theory, just solutions and ideas you can apply right away
DSL Engineering is instead the right book for you if you want to learn about advanced patterns and design decisions you should take when building a Domain Specific Language

4. ACKNOWLEDGMENTS AND SOURCES

Acknowledgments

We would like to thank Markus Völter for providing valuable feedback and improvements to this document.

Sources

[1] MPS Community Meetup 2018 - Challenges of the Dutch Tax and Customs Administration (DTCA) https://www.youtube.com/watch?v=_-XMjfz3RcU

[2] Domain-specific languages to implement Dutch tax legislation and process changes of that legislation. https://resources.jetbrains.com/storage/products/mps/docs/MPS_DTO_Case_Study.pdf

[3] A Domain-Specific Language for Payroll Calculations: an Experience Report from DATEV Markus Voelter, Sergej Koscejev, Marcel Riedel, Anna Deitsch and Andreas Hinkelmann

https://voelter.de/data/pub/voelterEtAl-payrollDSL.pdf

[4] A DSL for Payroll Calculations by DATEV, by Marcel Riedel

https://www.youtube.com/watch?v=uof9ERpBXSk

[5] Domain-specific modelling for the Dutch Tax Agency: how MDE enabled civil servants to program
[5] Domain-specific modelling for the Dutch Tax

Agency: how MDE enabled civil servants to program tax law. https://mde-network.com/wp-content/uploads/2021/10/A-success-story-Dutch-Tax-Agency-final.pdf

[6] MDSE at the Dutch Tax and Customs Administration, by Barbara Adema

https://staf2019.win.tue.nl/wp-content/uploads/2019/09/STAF-ID-Adema.pdf

[7] Software maken met een fabriek https://werken.belastingdienst.nl/nieuws-en-artikelen/software-maken-met-een-fabriek-115

[8] Regelbeheer https://gitlab.com/commonground/virtueel-inkomstenloket/regelbeheer

5. NEXT STEPS

If you have found this interesting and would like to explore MDE further, there is help to hand.

Join us in MDENet (www.mde-network.org) to engage with an international network of experts in model-driven engineering, access learning resources, events, and funding.

ABOUT THE AUTHORS

Federico Tomassetti is a Language Architect at Strumenta, a boutique consulting studio he co-founded. In his role at Strumenta, Federico is involved in different Language Engineering projects, ranging from the definition of Domain Specific Languages for different domains to the design of transpilers, editors, and interpreters. He got his PhD in Language Engineering between Italy and Germany. He speaks regularly at conferences and organizes the Strumenta Community, to hold discussions around Language Engineering.

Steffen Zschaler is a Reader in Software Engineering at King's College London and the director of MDENet, the expert network on model-driven engineering. His research interests include model-driven engineering, graph transformations, and principled simulation engineering. More information can be found at www.steffen-zschaler.de and he can be contacted at szschaler@acm.org.

Meinte Boersma is an active practitioner of model-driven software development and DSL engineering since 2007. He designs, implements, and deploys DSLs for various companies and organizations, using a multitude of technologies. Furthermore, he speaks on conferences, publishes blogs, participates in the development of open source standards and frameworks for DSL engineering, and is actively involved with the software language/DSL engineering community.

ABOUT MDE NET

The EPSRC network MDENet brings together research and practice in Model-Driven Engineering (MDE). We will do this by:

Driving Future Research

We will establishing a clear understanding, shared by the community, of the current state of the art in research and the challenges at the forefront of academic research and industrial use. We also aim to identify and create opportunities for cross-disciplinary MDE research.

Training the IT Industry

We will curate and produce training material, lowering the barriers to entry for potential new users of MDE technology. These, in turn, will bring new demand and challenges for future research and development in MDE, creating demand and support for new and improved training materials.

Building the MDE Brand

Focusing on activities to increase the visibility of MDE research, and on community building. MDENet will be the home of the MDE community and the authority on MDE, opening this space to individuals and teams not previously involved in MDE

This network is a broad church. If you are interested in software development, automation, or (computational) modelling in other domains (biology, AI, robotics, finance, ...), this network will likely have something for you.

Join our community platform