DELIVERABLE

D3.5 SD Tool Initial Prototype

Project Acronym:	PoliRural	
Project title:	Future Oriented Collaborative Policy Development for Rural Areas and People	
Grant Agreement No.	818496	
Website:	www.polirural.eu	
Contact:	info@polirural.eu	
Version:	1.0	
Date:	22 March 2021	
Responsible Partner:	22SISTEMA	
Contributing	Avinet	
Reviewers:	Anne Gobin (Vito) Karel Panek (Nuvit)	
Dissemination Level:	Public	X
	Confidential - only consortium members and European Commission	

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 818496

Revision History

Revision	Date	Author	Organization	Description
0.1	29/03/2021	Anne Gobin	VITO	Several improvements and corrections where introduced.

Table of Contents

List of Tables

List of Figures

Glossary

Term	Definition
API	Application Programing Interface
BOT	Behaviour Over Time
BQM	Base qualitative model
C	A programming language
CLD	Causal Loop Diagram
CPU	Central processing unit
Docker	A platform for containerized applications easing installation and deployment of applications in cloud environments
DSS	Decision Support System
Flask	A web service framework for Python
Git	Source code management system
GUI	Graphic User Interface
Http(s)	(Secure) Hyper Text Transfer protocol
KPI	Key Performance Indicator
LCM	Local customized models
OS	1 Operating System, i.e. Windows, Linux etc. 2 Open Source, i.e. a permissive license
Python	A programming language
Repository	A shared, managed storage and change management system used for sharing application code and XMILE models
SD - SDM	System Dynamics - System Dynamics Modelling
SSL	Secure socket layer
TRL	Technology Readiness Levels
UX	Interaction design (end-user centric application design)
WAI	Web Accessibility Initiative
WCAG	Web Content Accessibility Guidelines
WP	Work package
XMILE	OASIS XML Interchange Language (XMILE) for System Dynamics

Executive Summary

One of the key points in WP5 revolves around SDM modeling as a tool to support foresight. Apart from the specific issues raised in WP5, WP3 is also crucial for the usefulness of the SDM. This document is not about building the operative SD model, but translating and simplifying the technical and conceptual complexity to a level that can be managed by local agents and policy makers. This work started in D3.3 and is to be finished in D3.6, with the important milestone of the prototyping contained in the current deliverable.

The base model for the prototype, PoliRural SDM ed.3, is explained in chapter 3, together with the context of the foresight exercise.

Chapter 4 deals with the technical process to build the tool, including the realization of concept as explained in D3.3, preprocessing, and execution via web API. The chapter ends with an example to demonstrate how the developed API can be integrated into a HTML5 and Javascript application.

Since the beginning of the project the necessity has been signalled to engage a community of stakeholders with a vested interest in following, contributing to and furthering the work done on PoliRural with regards to System Dynamics. To help achieve this goal, point 5 describes the form of a database of local dynamics.

Licensing and distribution matters are covered in chapter 6.

The section on conclusion and next steps contains a description of the challenges ahead, including functional, organizational, technical and accessibility issues. The document ends up with some considerations about the business sustainability and planning.

1 Introduction

Deliverable 3.3 was the starting point in one of the main technological challenges PoliRural faces. This is not so much building an operative SD model, but translating and simplifying the technical and conceptual complexity to a level that can be managed by local agents and policy makers.

In the mentioned deliverable, the ground was put for such a development, both in technical and operative terms. This deliverable represents a further step in the challenge set out above. It is now time to build a prototype that can answer the needs and solve the difficulties encountered.

As was the case in D3.3, technical and operational aspects will be treated here, trying to cover the whole field of SDM for rural development.

This is a prototype, so it is important to emphasize the test aspects around it. Technical as well as operative assumptions are made. These will have to be evaluated by users, and so modifications may appear in later versions of the modelling tool.

In this sense, the function of the prototype is to state working hypotheses, and then draw conclusions, which may include changing some of the assumptions made.

1.1 Definitions and scope

As it is written in the Grant Agreement, this deliverable corresponds to Task 3.4 objective which is to oversee “the development work leading to the creation of a functioning OS simulation framework which can accept PoliRural models developed for and validated by the pilots”.

The model used for the development of the prototype is PoliRural SDM edition 3. This model contains all the evolution undertaken from edition 1, and it is also the basis for the 12 pilot local adaptation which is currently taking place. The model is the result of the works in T5.1 (model development as such), and T5.2, literature review. The model has been built considering both general literature and local specificities and policies, as described in D5.3.

T5.3 refers to regional futures: exploration and validation. This is a task to be completed in month 28 (September 2021), but some considerations from the point of view of exploration and validation have also influenced the design of the prototype and the workflow to use it locally in the different pilot regions.

Even though in the Grant Agreement is written “migration of a well-defined model is a relatively trivial task compared to iterative and experimental programming of models, allowing for the main emphasis of the project to be kept on policy issues rather than software development” it has to be said that the technical development is not a trivial task, especially when considering the effort of building understandable and useful interfaces, and also designing comprehensible workflows compatible with a non-SD specialists.

Having said that, it is true that a good deal of the discussions and designing of the prototype have turned around the subject raised above: the need to facilitate understanding and access to a population not necessarily introduced to the concepts SDM considers.

1.2 Objectives, purpose

With the antecedents stated above, the objectives of this deliverable are three, namely to:

- Build a working prototype according to the considerations and constraints explained in D3.3.
- Fix the functioning workflow for such a solution to be practical and useful for local authorities.
- Prepare the solution to be tested and improved in the final product.

2 PoliRural SDM

At the core of the prototype development is the SDM, and more specifically PoliRural SDM edition 3. This model is referred to as PoliRural Base Qualitative Model in D3.3 (point 5.1).

The complement to the main model is the Local Customized Model (point 5.2 in D3.3), and it expresses the possibility of adding new dynamics reflecting with more detail local realities. In this case the prototype deals with the construction of the GIT repository (see chapter 5) for the local dynamics database and the workflow needed to make it possible.

SDM is a tool to help the foresight exercise by rural areas. This is why there is a need to understand the role of the model in the foresight and what are the specifications in this context.

2.1 Description of the model

What follows is a brief description of the PoliRural Base Qualitative Model. Additionally, a reference of the works to produce Local Customized Model has also been introduced. Together they constitute the basis of the prototype and the GIT repository.

2.1.1 PoliRural Base Qualitative model

The Base Qualitative Model is PoliRural SDM ed. 3, and it has been evolving from the first edition. The idea of the model is to be a general template from where to adapt to local data and dynamics.

The model is made up of eight modules: POPULATION, EDUCATION, QUALITY OF LIFE, AGRICULTURE, NATURAL CAPITAL, EMPLOYMENT, RURAL ATTRACTIVENESS and RURAL RETENTION CAPACITY. They are highly integrated with each other as shown in the image below, and together they reproduce the main dynamics of a generic rural area in Europe. Deliverable D5.3 contains a detailed description of the model.

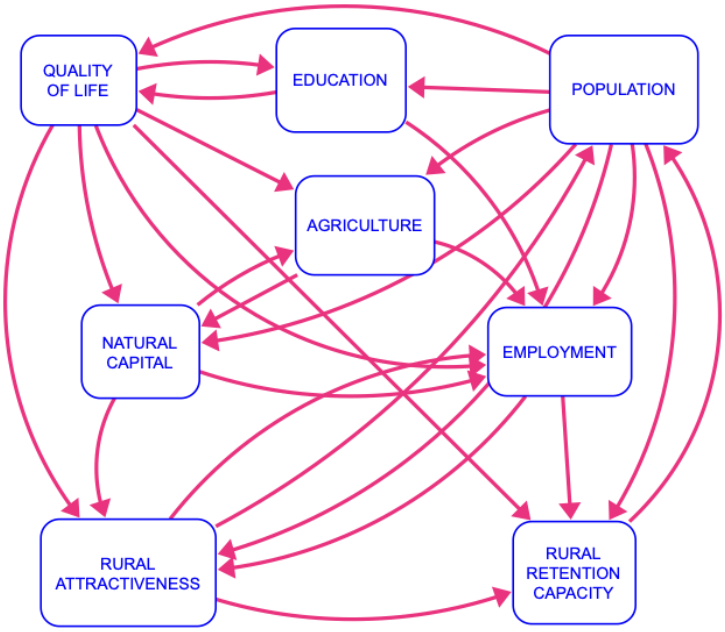

Figure 1 SDM ed. 3 Modules' Structure

The main challenge when building the model has been to find an equilibrium between relevance, usefulness and resources availability. The result is a complete working model in which the value is not so much in the loops described but in putting together single relations between variables.

Then there is a first value as a visual map of relations. From there, in a further step, the model contains a global consideration of known and quantified facts or relations, and some others also known but not quantifiable. In this sense the value comes from putting all known relations and facts in a single model.

Complexity is thus tackled partially. The model reflects the complexity of multiple relations and agents acting iteratively over time. At this level complexity is considered. But on the other hand, complex systems are highly dependent on initial conditions. This mathematical complexity is not reflected in the model. This would require an in-depth analysis in each of the pilot areas and a calibration effort out the scope of the project. Therefore local implementation in the pilot regions focuses on the most important relations between the different modules.

The discussion about the usefulness of the exercise follows in 3.2, dealing with SDM in the context of foresight.

2.1.2 Local Customized Model

The local customized models' objective is to develop the model further and get more accurate results and interpretations of local reality and perspectives, as stated in D3.3.

In this sense, local customized models may have the form of new modules or new dynamics added to existing ones. The origin may be a particularly dominant local dynamic not observed in the template, or a strong impact to consider in the scenarios for the future.

These pieces of model (local rural dynamics) can be expressed in SD feedback loops or even in causal loop diagrams (CLD).

Whatever is the case, the prototype includes a procedure to introduce them into the GIT repository, conveniently tagged and classified, and with a user interface so that it becomes a source of knowledge, comparison and best practices for rural communities.

2.2 SDM in the context of foresight exercise

SDM is thought of as a support tool within the foresight analysis. It is important to again make clear what SDM can and can't offer to the exercise.

Modelling represents a step beyond the literal description, because it confronts local agents to think in terms of the whole picture: how general trends can affect this or that module/dynamic? what indicator can better measure the impact of the foreseen trend? These are the type of questions that can arise when translating foresight reflexion into the model.

SDM is never about forecasting exact future values for a given indicator, but about studying possible future trends and scenarios. SDM is in this context an exercise to help local agents to think about the future in two ways, the effect that general trends may have at local level; and the way policy options may correct unwanted impacts.

SDM is going to help them in two ways: by forcing them to think in terms of the whole rural ecosystem and structuring to this view impacts and measures; and by analysing and comparing trends that result from combining different policy options and impacts.

3 SD Tool Construction

This portion of the deliverable describes the process of making a system dynamics model executable independent of system dynamics modelling software. The description is heavily targeted at technical users and may not be meaningful to readers who are not familiar with software development in programming languages like Python and JavaScript, web services and common web standards. This complexity, though regrettable from the point of view of readability, is necessary to provide the reference information other developers need in order to adopt, adapt and improve upon PoliRural results.

There are, as outlined initially, two separate aspects to validating the usability of SD in the domain of foresight analysis and spatial planning. Most importantly, there is the modelling challenge of gathering and transforming real-world user requirements into a functional model.

Secondly, there is the challenge of making the model available to users who do not have expert knowledge in system dynamics or access to the professional software suites and concepts used in that domain.

The approach to solving the second challenge has followed an entirely model-agnostic manner methodology. Thus, the results of PoliRural should form a suitable starting point for automatically making any system dynamics model executable as a web service and callable from any client application.

That may offer significant benefits in PoliRural in terms of being able to evolve models throughout the project, and for authoring additional, simple models that can plug into standalone decision support systems already used in foresight and planning processes. The same component may also be used for SD models in other domains—both overlapping, adjacent and detached.

Simulation and forecast is not new to neither planners nor foresight analysts. The added value from SD is that this technology permits the encoding of more complex models that keep track of a greater number of concurrent variables, as well as the ability to consider feedback effects over time.

This increased capacity does, however, come at a price: increased complexity. The complexity of the models themselves have been discussed at length in chapter 3 above, the other complexity is the introduction of a technical barrier.

There are several dimensions to this barrier:

1. Domain specific terminology
2. Abstract modelling language, notation
3. Specialized software
4. Expensive licensing costs

The purpose of the SD-tool experiment is to break down these barriers from the perspective of non-SD experts who wish to use simulations as a supporting tool within their own expert domains.

PoliRural takes as a starting point that developers of existing decision support systems that are being used by planners will implement clients that call the API and run the simulations. To demonstrate how this may work, a demo application is provided and made available via the Digital Innovation Hub.

To achieve the potential benefits of the automatic production line from models authored in professional SD-tools to open Web APIs, it is necessary to integrate these APIs into existing decision support systems. That way their existing users get introduced to SD capabilities through an application that they may already be using and a terminology with which they are familiar.

In the following sections, we described how we have executed the incremental steps identified in D3.3, what obstacles we have come across, how we have mitigated them and what we consider potential for further improvements.

3.1 Realization of concept from D3.3

This section describes in detail the steps required to make an SD-model executable outside of professional SD software. The starting point for the process is a XMILE-file, an open XML-based interchange format for SD-models.

3.1.1 Setup of environment for server application

The tool chain uses Python to migrate the model. There are many Python distributions, but for ease of setup and configuration, this walkthrough assumes the use of Anaconda or Miniconda due to the maturity of their associated package managers. Since the deprecation of Python 2.x, the toolchain is based on Python 3.x, at the time of writing version 3.8.

Therefore, the first step is installation of the Anaconda environment on the host (and/or development machine).

```
# Download Anaconda from: https://www.anaconda.com/
# Choose either
# - 64-Bit Graphical Installer (457 MB)
# - 32-Bit Graphical Installer (403 MB)
# Downloads exist for Windows, MacOS and Linux
```

Contemporary development and integration relies heavily on package managers and ready-to-use package-repositories that bundle programming libraries that extend the capabilities of the basic programming language.

In Python, the default package manager is PIP (which rather unhelpfully resolves recursively to PIP Installs Packages). While PIP provides access to a broad range of libraries, compatibility between them must be managed. The Anacondas package repository contains many packages that are ready to use for production environments and that are targeting specific suites of tools.

Having done that, the next step is to create a suitable environment, this is done as such:

```
# Create a new environment
conda create --name PoliRural

# Or create it with a config file
conda create --name PoliRural --file requirements.txt

# Activate the environment
conda activate PoliRural
```

Next it is necessary to create a base directory for our application, typically, this would be done by cloning the development repository.

```
# Open a terminal and enter the development directory
# E.g. /var/lib/development
cd /var/lib/development

# Cone the PoliRural repository for the server application
git clone https://gitlab.com/PoliRural/pysd-PoliRural-server.git

# Enter the development catalogue
cd pysd-PoliRural-server
```

Next it is necessary to install the relevant libraries to support our application.

```
# If you created the conda environment with a config file,
# all packages will already be installed, otherwise, you may:

# Install packages one-by-one manually
conda install pysd
conda install Flask
conda install Flask-Cors

# Or install all packages in one go using PIP and the supplied
# requirements.txt file from the cloned repository
pip install -r requirements.txt
```

The environment is now configured and ready to use. Test it using the command:

```
# Execute the script test.py
python test.py

# If everything is ok, this should output:
Everything works!
```

3.1.2 Setup environment for the demonstration client application

The demonstration client is implemented in HTML5 and Javascript using the well-proven and robust React framework, developed and notably used by Facebook to power their main application.

The client demonstrates the possibility of executing system dynamics models using open source execution engines, outside of expert systems. The most important part of the application is the client-side compliment to the Web Service API described above.

Anyone wishing to integrate SDM execution capabilities into a third-party application needs to connect to the Web Service API. The demo client provides a boiler-plate template for creating such applications and includes a client-side library to communicate with the web services using Javascript xHr-requests by means of the underlying Superagent http-request-library.

To check out a local development version of the SD demo application, do the following:

```
# Open a terminal and enter the development directory
# E.g. /var/lib/development
cd /var/lib/development

# Cone the PoliRural repository for the system dynamics demo application
git clone https://gitlab.com/PoliRural/sd-demo-application.git

# Enter the development catalogue
cd sd-demo-application
```

Having entered the development directory, you can now install the dependencies and run the application using either yarn or npm.

```
# To use the Node package manager:
npm install

# or to use Yarn:
yarn install
```

3.2 Pre-processing

At this stage, we have a running application capable of executing an SD-model in a back-end Web Service API documented and published according to the OpenAPI (aka Swagger) standard.

However, users might wish to replace the model being executed with a model version of their own. This can be done by editing the model in Stella Architect, or models can be edited in free or freemium editors such as Vensim, that have licensing models that are more inclusive of academia, government and small businesses.

The Web Service API application comes with a converter application that accepts as input either qualified XMILE-files or Vensim-files.These are processed and transformed into executable Python code for each auxiliary, variable, valve, stock and flow in the designed models.

To convert a model, make sure that it is saved on your computer and navigate to the utilities folder in the server application described in section 3.1.1 above.

```
# Step into the development directory
cd %development%
cd pysd-PoliRural-server
cd utilities

# Execute the converter utility
python converter.py --input /home/user/my_model.xmile --output
/home/user/my_model.py --debug
```

There are a couple of caveats to this process.

3.2.1 Non-conformant XMILE

XMILE is a standard with rather fewer applications than might be desirable. It acts as a theoretical vessel for transporting models between execution environments and software packages, but the lack of a range of qualified tools where it can be edited means that it is often imported and exported from the same tools, e.g., Stella.

This means that certain idiosyncrasies in the way the standard is implemented as an export format in Stella will go undetected as the model often returns to Stella by means of the same way which it came out.

One such issue is the naming of stocks, auxiliaries, variables etc. Where descriptive names are used in Stella Architect stock-flow diagrams, these are translated into XML-safe strings by the export routine. What the routine fails to do is apply identical normalization to source and target references, meaning that some flows may be broken.

To mitigate this, an utility has been made to clean up models and report any remaining issue with intelligible feedback from the cleaning software.

```
# Execute the cleaning utility prior to converting the model
python repair.py --input /home/user/my_model.xmile --output
/home/user/repaired.xmile --debug
```

3.2.2 Functional limitations

The execution engine is a bit more limited in its implementation of “functions'' than what is the case in the expert software. Below is a list of the principal functions that are implemented in Pysd and their names as used in Vensim and XMILE along with the corresponding function in Python. Functions that the model is not capable of translating will have to be wired up “by hand”. This is not difficult, but requires the user to familiarize with the Pysd source code.

Vensim/XMILE function	Python Translation
COS	np.cos
EXP	np.exp
MIN	min

<=	<=
STEP	functions.step
PULSE	functions.pulse
POISSON	np.random.poisson
EXPRND	np.random.exponential
SIN	np.sin
>=	>=
IF THEN ELSE	functions.if_then_else
LN	np.log
PULSE TRAIN	functions.pulse_train
RAMP	functions.ramp
INTEGER	int
TAN	np.tan
PI	np.pi
=	==
<	<
>	>
MODULO	np.mod
ARCSIN	np.arcsin
ABS	abs
^	**
LOGNORMAL	np.random.lognormal
MAX	max
SQRT	np.sqrt
ARCTAN	np.arctan
ARCCOS	np.arccos
RANDOM NORMAL	self.functions.bounded_normal
RANDOM UNIFORM	np.random.rand
DELAY1	functions.Delay
DELAY3	functions.Delay
DELAY N	functions.Delay
SMOOTH3I	functions.Smooth
SMOOTH3	functions.Smooth
SMOOTH N	functions.Smooth
SMOOTH	functions.Smooth
INITIAL	functions.Initial
XIDZ	functions.XIDZ
ZIDZ	functions.XIDZ
VMIN	functions.vmin
VMAX	functions.vmax
SUM	functions.sum
PROD	functions.prod

GET XLS DATA	external.ExtData
GET DIRECT DATA	external.ExtData
GET XLS LOOKUPS	external.ExtLookup
GET DIRECT LOOKUPS	external.ExtLookup
GET XLS CONSTANTS	external.ExtConstant
GET DIRECT CONSTANTS	external.ExtConstant
GET XLS SUBSCRIPT	external.ExtSubscript
GET DIRECT SUBSCRIPT	external.ExtSubscript

Table 1 Python translation of XMILE functions

3.2.3 Model compilation and execution time

Execution in a web environment poses another interesting challenge compared to that of working on the local desktop. CPU-cycles and memory are generally more plentiful on a web server, but they are also shared by concurrent users.

On a local computer there are no competing users and if executing a model should require 100% of the CPU-time for some duration, that is acceptable. On a server, it is necessary to tune the scripts so that the memory consumption and CPU usage of each does not “break” the server for other concurrent users.

In the upcoming period, until the release of D3.6, further benchmarks will be done, but at present, the application has been tuned to allow up to 50 concurrent users, i.e. 50 executions of the model within the same second. This is accomplished from a server with 8 CPU cores, 16GB of RAM and running behind Apache Httpd server.

Web applications also are expected to respond in real-time with acceptable lag being no more than <= 2 seconds. Initially, we envisaged to recompile the model from XMILE to Python during the request itself, so as to be able to ALWAYS keep the model up to date. However, this turned out to be too slow as the model conversion accounted for more than 3 seconds out of each execution whereas the model execution took no more than 800ms.

Thus, we decided to separate the model conversion into a separate web service and cache the converted models so they could be reused between each execution of the API. Seen against the challenges of standards compliance mentioned in chapter 7 below, this caching might be a necessary step in any event. Introducing a new XMILE-file might inadvertently break the conversion process and thus render the model in a state where it is not possible to execute.

3.3 Execution via web API

The Web API has for the present been given the working title “PoliRural System Dynamics API” and is exposed at a public URL-endpoint, meaning that anyone can make requests to it from anywhere, as long as they have Internet connectivity.

Since Pysd, the basic component that the process rests on is developed in Python, the implementation of the API has also been made in Python, using the Flask and flask-restx libraries.

These libraries simplify the process of establishing a secure and robust http REST-api including versioning, authentication/user tokens and all common web API features. Furthermore, flask-

restx provides a syntax for inline code documentation that is used to auto-generate OpenAPI end-user documentation to aid parties seeking to adopt the API.

OpenAPI defines itself as follows:

> "*The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.*
>
> *An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases.*"

And it is this latter capacity that the PoliRural System Dynamics API harnesses as it automatically generates a browsable API-documentation web site with data model documentation and the capacity to run tests from within the browser, not requiring any other tools.

When pointing the browser to the public URL-endpoint where the API is hosted, the user will be presented by a screen that introduces the PoliRural System Dynamics API. On this page, there will be a short description of the API as well as information about the version. D3.5 signifies the release of the 3rd version of the PoliRural SDM, as developed by 22Sistema, but the 1st version of the API, developed by Asplan Viak.

There will be different end-points for each API version to ensure backward compatibility and not cause breaking features for any application that might be integrated with the API. This is not likely to be an issue during the PoliRural project life-time, but is nonetheless a reasonable precaution.

What a web service API can do is expressed through methods. These have their own URL and accept one of the http verbs GET, POST, PUT or DELETE. A method can receive input data either as part of an embedded pattern in the URL used to call it for GET-requests, or input data can be provided as part of the http message body for POST-requests.

Methods can be grouped into "namespaces" that share the same root URL path. In this API, the methods have been grouped according to the SDM version, under a namespace called PoliRural System Dynamics API.

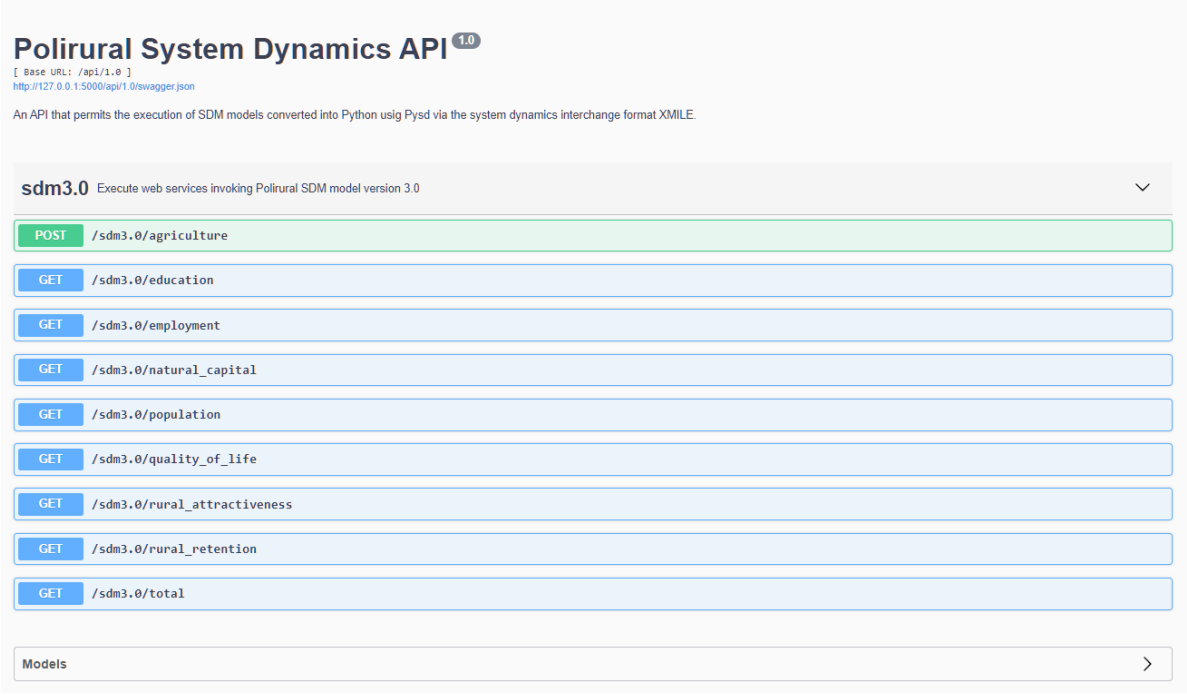

Figure 2 Automatically generated Open API documentation page for PoliRural API

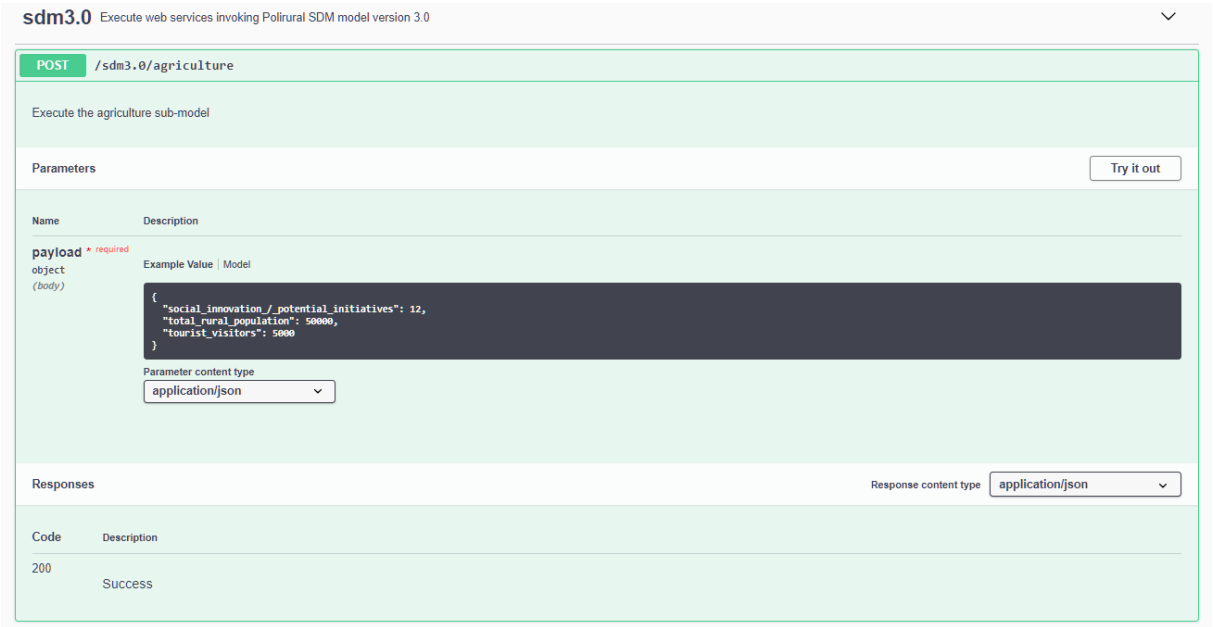

Figure 3 Detailed information about each method exposed through the API

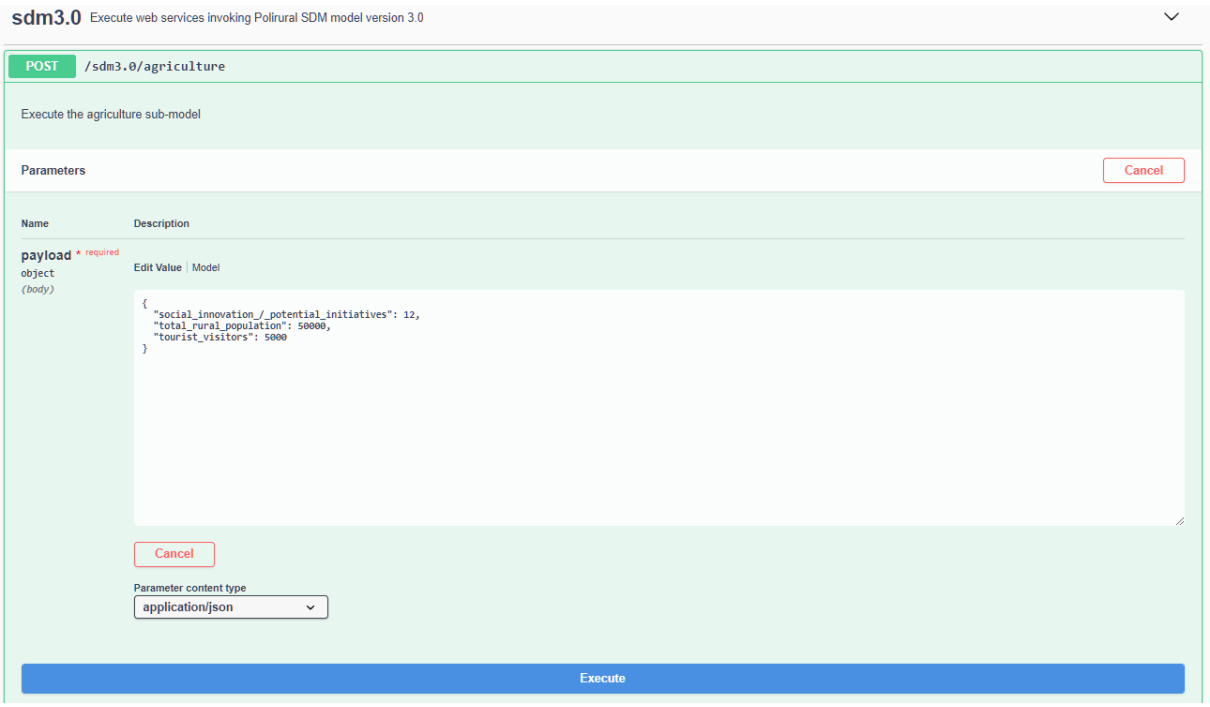

Figure 4 The Open API visualization engine Swagger-UI allows users to test web services directly from the browser

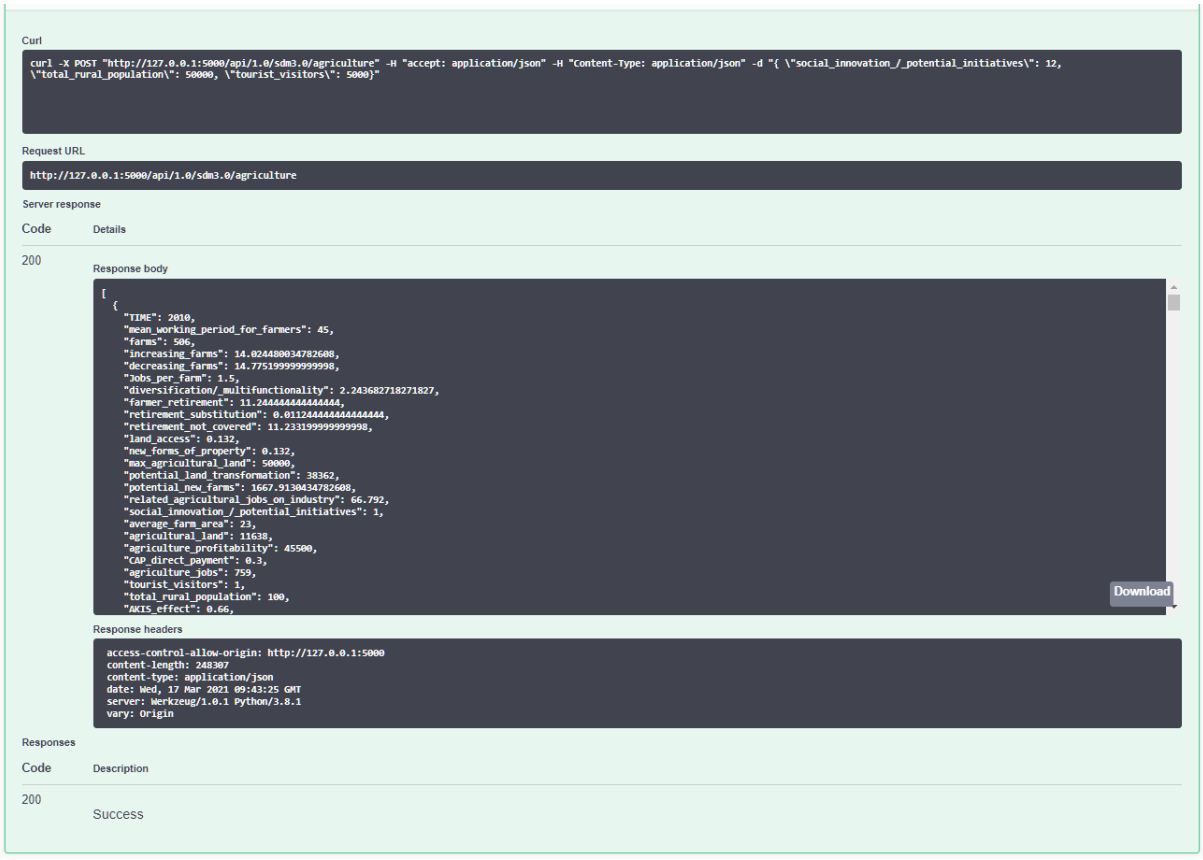

Figure 5 The responses from the web services shows parsed, human-readable results that make it easy for developers to understand the requests and responses that are required to interact with the web service API

3.4 Example end-user application to demonstrate API

While the production line from proprietary authoring-tool via XMILE and Pysd to an Open API for SDM-execution is the most important result of this activity, such an API serves little purpose unless integrated into an application.

That is where the System Dynamics Demo Application comes in. In its first version, it is meant to demonstrate how the API can be integrated into a HTML5 and Javascript application that is independent of proprietary expert software such as Stella Architect and conditionally free publishing platforms such as Isee Exchange.

There are four key elements that the demo application seeks to illustrate:

1. A stand-alone graphical user interface that doesn't have to look like the typical work area of a system dynamics modelling tool. A user is interested in simulation results rather than having to learn the domain-specific terminology of SDM or its graphical artifacts.
2. A model with many parameters can be broken down into steps that can be completed using a "wizard" paradigm where each parameter is introduced in human readable text and specified prior to going to the next step, all the way until completion.
3. A wizard does not mean that all the bells and whistles of a model have to be hidden under a naive and simplistic UX-trick, on completion of the wizards, or as an alternative to them, users can manipulate all model parameters prior to executing the models.
4. Exploration of the results can be done in a multi-dimensional fashion where the time-series are visualized in a series of interlinked charts. By setting filters in each chart, intersecting dimensions and their associated charts are also updated.

The demo application will be linked from the PoliRural Digital Innovation Hub along with links to the Web Service API and the relevant code-repositories.

4 GIT Repository – Local Dynamics Database

PoliRural is piloting the value proposition of employing System Dynamics for rural policy development, planning and foresight analysis. But the activities within the project are not sufficient on their own to ensure sustainability of the results and form the foundation for a sustainable market.

Therefore it is necessary to engage with a community of stakeholders with a vested interest in following, contributing to and furthering the work done in PoliRural with regards to System Dynamics.

To conceive such a community and grow it into something that is self-sufficient is not a trivial task and involves both a limited technical component as well as a more comprehensive outreach and communication activity. This will be closely coordinated with the overall dissemination and sustainability efforts of PoliRural.

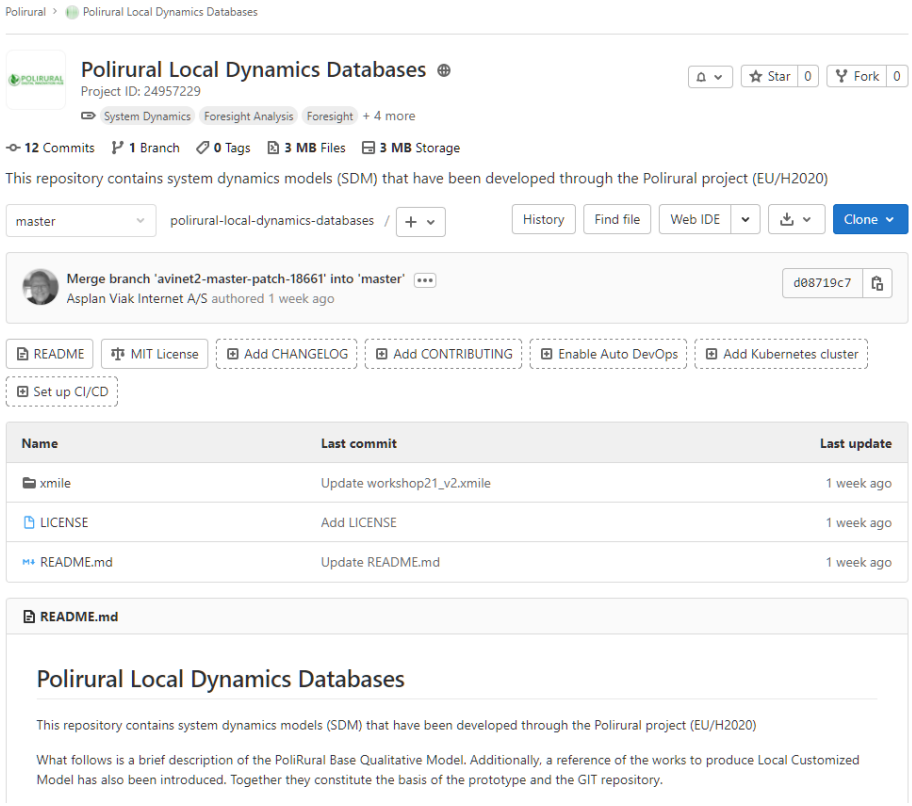

Figure 6 Online Repository for PoliRural Local Dynamics Databases

An essential component in enabling collaborative assessment, enhancements and quality improvement of shared models is a place where they can be accessed, modified and versioned.

Since we are dealing with an open text-based exchange format that is not platform specific, i.e. XMILE, we have chosen to establish a public Git repository on Gitlab that we will use as a vessel for sharing and versioning our models.

The repository is available on the following URL:
https://gitlab.com/PoliRural/PoliRural-local-dynamics-databases

As for the development of the community, we are presently in the process of identifying relevant existing networks that we can 'tap into' in order to build further on existing initiatives. We believe this strategy to have a higher chance of success than trying to establish a new community from scratch.

In any case, the objective of the effort is to ensure that there will be stakeholders, in addition to the PoliRural pilots and technical partners, who have a continued interest in the project results on the day when the project is finished and who can sustain the models until a market can be developed and a sound business model that serves the established market.

5 Licensing, availability and distribution

The experience so far in PoliRural suggests that system dynamics modelling in rural development is very sensitive to local conditions and that it is difficult to conceive abstract, pan-European models that provide output that is interesting at a local level.

For this reason, the value-proposition of PoliRural is not the shared models in their unmodified form - but the derived, local customized models that are adapted to business case and region.

To release this value-proposition, it is necessary that the base qualitative model, and such models as may be authored throughout PoliRural will be made available as templates that others can continue working on.

There is no envisioned down-side to this as the business potential rests on consultancy services for training, modelling and software integration, and since the market as it looks now is too fragmented to be serviced on a large-scale.

Inheriting from the software industry, we have chosen to release models and software under an MIT-like license. This protects the copyright, interests and liabilities of the developers at the same time as it grants unlimited exploitation rights to any third party.

```
Begin license text.
Copyright <YEAR> <COPYRIGHT HOLDER>

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to permit
persons to whom the Software is furnished to do so, subject to the
following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN
NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE
USE OR OTHER DEALINGS IN THE SOFTWARE.

End license text.
```

6 Conclusion and next steps

The central hypothesis that all SD-work in PoliRural must consider is whether system dynamics modelling adds tangible value to rural planning and foresight analysis. It is not a foregone conclusion that this is so. However, early results and experiences point to the usefulness of the technique.

Thus far there is evidence that simulation is an interesting tool without taking into the equation the costs of adopting it to the beneficiaries. For the upcoming period, i.e. the interval between this deliverable and the release of the final version of SDM ed. 4 will need to look particularly at how and whether system dynamics efficiently and economically can be adopted by rural planners and foresight analysts, under which circumstances and on what conditions.

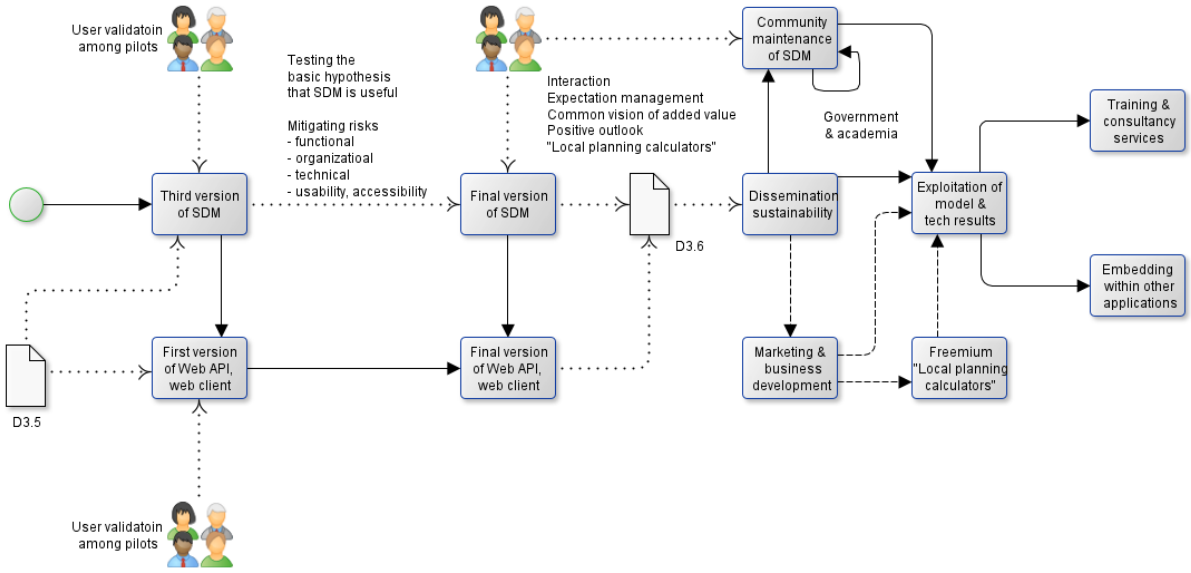

Figure 7 SDM tool in the context of PoliRural project

6.1 Challenges and issues to be mitigated

There are a number of issues to be observed and mitigated in the interim between the release of D3.5/SDM ed.3 and D3.6/SDM ed.4. These concern the rigorous testing of the basic presumption of PoliRural, namely that SDM offers a sustainable value proposition for rural policy development, planning and foresight analysis. But issues also include function, organizational, technical and usability issues, all of which are briefly commented upon below as they look at the time of completion of this deliverable.

6.1.1 Functional challenges

Rural areas are complex ecosystems, and so any approach considering multiple interactions and views will represent an improvement in the understanding and the impact of policy actions and measures taken. Systems thinking as a methodology and SDM as a tool deepen in this direction and local authorities and agents must incorporate this dimension to the analysis.

In the same direction as the point before is the need for indicators reflecting relations within the rural ecosystem and comparable for different regions. The need for such a view was evident in the conversations asking for information to the pilot areas. Despite the fact that much of that data was replaced by good proxies, appropriate indicators will reflect local dynamics more accurately.

The same can be said about indicators reflecting general trends. Rather than general ones, indicators that reflect the impact of these global trends on local dynamics will be of interest. So, instead of talking about climate change, CAP reform, neo-ruralization or local food chains, local translations of these trends are needed. This implies collaboration between general scientific production and local experts, producing information in the form of ecosystemic indicators reflecting the impact of global trends locally.

It has been clear from the beginning of the project, that the SDM exercise does not aim at a precision forecast of the future, but rather an understanding of the structures that give place to the observed trends, and the evolution of these in the future. In this sense this is exploratory modelling: the use of computational experiments to analyze complex and uncertain issues.

6.1.2 Organizational challenges

On the heels of the functional challenges, an observation that has been made in PoliRural, to date, is that modelling of local contexts requires local knowledge, and would greatly benefit from being customized by someone in the respective local government.

This stands somewhat in contrast to the current adoption of SDM that remains a relatively narrow expert-discipline that predominantly is the domain of large industrial, academia, and to the extent that it is used in government, by large national bodies or major urban municipalities.

Nonetheless, local dynamics must be reflected in the models to support local planning and analysis. A single, centralized model would be either too abstract or too complex to be of any service whatsoever. Thus, it would be beneficial if System Dynamics modelling was a discipline that was available in the organizations, on the same level as other ‘standard’ IT-tools like word processors, spreadsheets etc.

That is presently not the case. Nor is it likely that the economic climate in rural municipalities will give room to prioritize keeping this sort of skill-set in-house, given the wide range of tasks that they are obliged to by law. Thus, for the foreseeable future, we are left with a skills-gap that is not likely to aid in the uptake of PoliRural services.

Mitigating this challenge may be done by developing a streamlined production line for customizing the base qualitative model on demand.

Given that all such customization would begin from the same starting point and the same concepts, it would be possible for SDM-modellers to do this a good deal more efficiently than if they were to start from scratch. This would in turn minimize costs, and might render SDM-services within the financial reach of rural governments that might see it as beneficial.

Growing that engagement will be a concern going forward, assuming our experiment demonstrates that SDM indeed does pose an attractive value-proposition for rural planning and foresight analysis.

6.1.3 Technical challenges

There are a few technical challenges, too. One being standards conformance. The domain of SDM is dominated by a few large commercial system vendors who are not eager to let go of their customers. Thus, as is the case in most standardization, the industry has no obvious incentive to encourage the use of interchange standards.

XMILE is however an established standard, but lacks implementations to put it to the test, or at any rate a critical mass of such implementations. Therefore, each vendor's implementation of the XMILE has what one might generously call ‘application profiles’, or more accurately, perhaps, non-compliance.

This makes it difficult for two systems who rely on XMILE for their exchange of information to “shake hands'' when data starts to pass back and forth. This issue was uncovered in our work with making XMILE models executable in Pysd, relying first on the limitations inherent in the export from Stella Architect, and then on the limitations related to the implementation of XMILE as a model format in Pysd. Neither were flawless, but nor were either of them beyond quick-fixes and workarounds. Thus, this is not a show-stopper, but an issue that it is necessary to be aware of.

It will be an issue to be kept under observation, going forward.

The other technical challenge is the lack of system dynamics modelling tools that at the same time are good, stable and affordable. As long as the authoring of models is limited by the entry cost into the SDM software ‘universe’ it will constitute a barrier to the experimentation and engagement by prospective SD users who might use it as part of their their arsenal/repertoire/toolbox but who are not dedicated for this sole purpose by profession.

Some tools, like Vensim and Simantics, make it possible to develop models in affordable or open source editors. Vensim does not output XMILE-data, but Pysd supports import of Vensim models nonetheless. Simantics support XMILE, though it takes quite a bit of tweaking to enable it. With this tool, robustness is the concern. While it seemingly has all the bells and whistles of its commercial counterparts, it is prone to crashes/hangs that requires patience to learn how to work around.

Our continuous monitoring of state-of-the-art in the domain will look especially for modelling environments that can mitigate this issue and in that way enable a broader range of users to adopt and experiment with models, generating more demand and engagement.

6.1.4 Usability, accessibility

A topic that has been all but absent to date is usability testing. In the work that has been done until now, including D3.5, we have respected the boundaries defined by the discipline of SDM for how a model should look and how the interfaces that run the model should look and behave.

This means that a lot of “tribal SDM-language” is exposed to users who command terminologies native to their own professions and might have neither the time nor the will to take on a new all-encompassing analytical paradigm in their professional view of the world.

Here, the web execution of models poses an interesting value proposition, as complexity can be hidden and reduced. In its extreme simplicity, a model published on the web can be reduced to a single input field, a run button and a screen showing the results.

While that level of simplification would be detrimental to what we wish to test in PoliRural, it shows that there is a potential to make some amendments to full-fledged SDM model-execution environments as they exist in the commercial tools.

In the interim leading up to D3.6, PoliRural will consult users on this matter and make a number of experiments that will be integrated into the web based demo application to serve as an inspiration and starting point for secondary integrations.

6.2 Towards sustainability and business planning

The sustainability of PoliRural results is concentrated in a dedicated work package, and will only be touched upon lightly in this deliverable. We only name and briefly describe three important measures that we will emphasize in our contribution towards that work package.

6.2.1 Maintaining the model

The model will need to be maintained and evolved. New policy instruments or major changes in the framework conditions must be reflected in the models if they are to continue serving as starting points for customization to local conditions.

To achieve this, we will try to grow an interest community as described above, and to garner sufficient interest that the models can be sustained for a time without relying on project financing or profits from sales.

This will buy us the figurative elbow room needed to develop a market and build a sustainable business model for the post-project exploitation of the SDM models, API and applications.

6.2.2 Developing the market

The pilot partners in PoliRural are excellent starting points for growing a market, but are not in themselves enough to guarantee widespread uptake and interest. Therefore, we need to start thinking carefully about how to bring this market to fruition.

We believe increased awareness of simulation as a methodology might be of benefit and envision the possible implementation of a number of simple ‘local planning calculators’ that illustrate and demonstrate, in a simple way, how system dynamics can be employed. These are not meant as ‘serious’ SDM-models but as technology teasers that will woo users and awaken dormant demand among users who are unaware of the possibilities.

6.2.3 Exploiting PoliRural results

The ultimate objective of our sustainability strategy is that PoliRural results will be brought forward into self-sustained services that do not rely on first-party subsidies or third-party funding, but that are capable of generating sufficient income to justify its upkeep.

We believe there are three key types of services that can generate business:

1. Training services to introduce planners and analysts to SDM and the PoliRural base qualitative models as well as examples on how they can be customized.
2. SDM-modelling services to customize models on behalf of customers.
3. Software integration services to build custom applications that execute SDM-models out of the proprietary system context.

The conclusions and next steps chapter outline what will happen from now and onwards and will partly be followed up in D3.6 and partly through deliverables in other WPs.