Bringing distributed software development to SD modelling with Vensim

Nicki Daniel Helfrich[1] (corresponding author), Wolfgang Schade[2]

1) Fraunhofer Institute Systems and Innovation Research (ISI)
Breslauer Strasse 48, 76139 Karlsruhe, Germany
Phone: +49 721 6809 364
Email: nicki.helfrich@isi.fraunhofer.de

2) Fraunhofer Institute Systems and Innovation Research (ISI)
Breslauer Strasse 48, 76139 Karlsruhe, Germany
Phone: +49 721 6809 353
Email: w.schade@isi.fraunhofer.de

Version 1 (2008-03-21)

Abstract

Maintaining one of the most complex System Dynamics model – ASTRA – with a group of more than 5 economists, we were facing two main problems. First, collaboration was difficult because all developers had to work with different files and changes had to be manually transferred into one model. Second, calibration was time consuming, since the complete model needs various minutes for only one run even on high end computers.

We found a solution to these problems in transferring techniques from distributed software development to SD modelling. We split our complex ASTRA model into more than 40 modules, developed standards for these modules to be able to run them independently and to enable automatic merging of any amount of modules to one model, developed a tool for automatically executing this merge in order to run the complete ASTRA model and we set up a version controlled repository accessible by all developers via internet to manage the simultaneous development work of our modelling team which is spread out over three institutes in two countries.

In this paper, we would like to present in detail the individual task as described above and conclude with our experiences after this major transformation of our ASTRA model.

Keywords:

Large scale SD modelling, modularization, version controlled repository, distributed modelling

1 Introduction

Maintaining one of the most complex System Dynamics model – ASTRA – with a group of more than 5 economists, we were facing two main problems.

First, collaboration was difficult because all developers had to work with different files and changes had to be manually transferred into one model. With this way of working, it was highly difficult or even impossible to maintain one up-to-date model containing all implemented structures by all model developers.

Second, calibration was time consuming, since the complete model needs various minutes for only one run even on high end computers. Calibrating the ASTRA model could only be done in reasonable time by sequentially extracting each logical unit of the model, creating a standalone version thereof, calibrating that part and putting the calibration results back into the complete ASTRA model as described by Schade and Krail (Schade, Krail 2006). This used to be a time-consuming task containing many repetitive steps.

We found a solution to these problems in transferring techniques from distributed software development to SD modelling. The underlying idea is that it should be easier to maintain many small modules rather than one big model.

Therefore we split the ASTRA model into more than 40 modules. These were designed in a way to make it possible for each module to be run as a standalone SD model, which is essential for testing during extending the functionality as well as for the calibration process.

Also, each module was designed in accordance with dedicated standards in order to enable an automated merging of any number of modules into one functioning SD model.

For performing this merge, we developed a java-based tool, the *Merger*, using "regular expressions" technology. It produces a valid SD model which can be run in Vensim without any further adjustments. This was necessary since Vensim does not support modularization of SD models. The development of *Merger* was inspired by the development of the conductor as described by Thompson and Bush (Thompson, Bush 2006). We would like to explicitly thank these developers for sharing their ideas with our community, enabling us to further develop their ideas to implement software which completely suits our needs.

Additionally to the modularization, we had to think of a way to systematically handle the problem of different versions resulting from the ongoing development of our model by a team spread out over three locations in two countries.

Therefore, we decided to introduce a "version control" software, namely *Subversion*, a widespread, open-source software. It allows a team of developers to always keep up with the current status of a project via a common file sharing server accessible via internet. About four hundred revisions of ASTRA later, our experience is highly positive and we never regretted our decision to go down that road.

This work was done during the TRIAS project as described by Schade et al. (Schade et al. 2008) and Krail et al. (Krail et al. 2007). This paper describes in detail all the steps taken.

Chapter 2 gives an overview on the original structure of ASTRA. Chapter 3 describes the standards needed for modularization along with the tool we developed for merging modules back to one model. Chapter 4 briefly sketches the technology used. Chapter 5 describes how we use a version controlled repository for team collaboration and chapter 6 gives a short overview of the practical work with the tool we developed before chapter 7 draws a final conclusion.

2 Where we came from – the structure of ASTRA

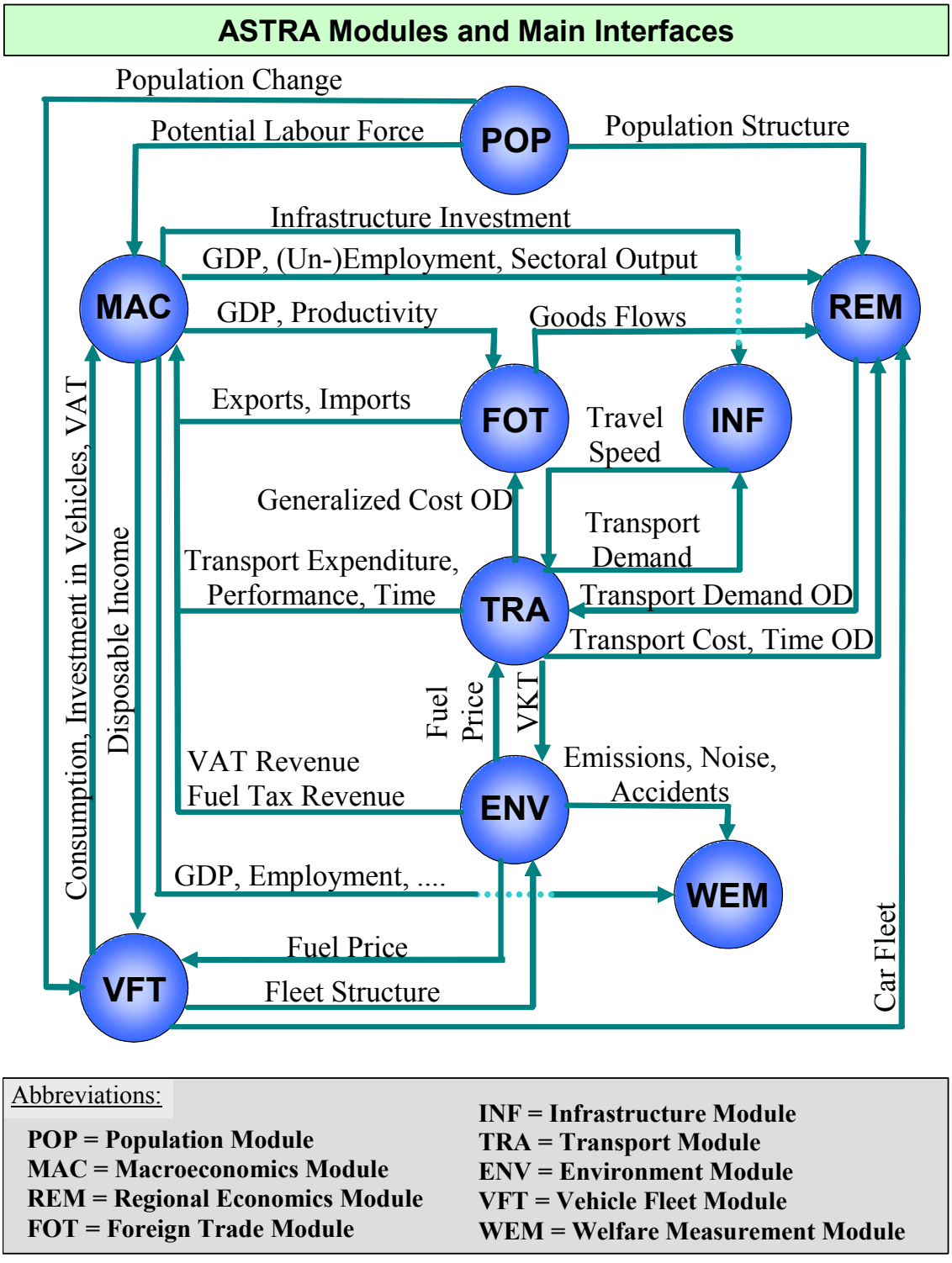
When starting the project described in this paper in early 2007, the complete ASTRA model was implemented in one single Vensim model file, leading to the problems described in the introduction chapter.

ASTRA (=Assessment of Transport Strategies) is a System Dynamics model generating time profiles of variables and indicators needed for policy assessment. Details of the ASTRA model are described by Schade (Schade 2005). Originally ASTRA was developed on the base of existing models that have been converted into a dynamic formulation feasible to be implemented in System Dynamics. Among these models have been macroeconomic models and classical four stage transport models (SCENES, ME&P 2000). ASTRA runs scenarios for the period 1990 until 2050 using the first twelve years for calibration of the model. Data for calibration stems from various sources with the bulk of data coming from the EUROSTAT (2005) and the OECD online databases (2005).

The ASTRA model consists of nine modules covering the 27 Western European Union countries (EU27) plus Norway and Switzerland, (EU29). The major interlinkages between the nine modules are shown in Figure 1. Purposes of the modules are:

- Population module (POP) calculates the population development and population structure for the EU29 countries with one-year age cohorts.
- Macroeconomics module (MAC) provides the national economic framework. The MAC combines different theoretical concepts as it incorporates neo-classical elements, Keynesian elements and elements of endogenous growth theory.
- Regional economics module (REM) describes spatial changes and the generation of transport on the level of sub-national functional zones.
- Foreign trade module (FOT) estimates trade flows by sector by country combination e.g. trade of vehicles from Sweden to Spain etc.
- Transport module (TRA) provides the modal-split of transport demand and calculates the transport performance by mode for passenger and freight transport as well as vehicle kilometres travelled.
- Vehicle fleet module (VFT) delivers the composition of the road vehicle fleets differentiated into different vehicle sizes, engine types and emission standards.
- Environment module (ENV) calculates the fuel consumption for the different fuels and the emissions of transport. Based on the fuel consumption also fuel tax revenues are calculated.

Figure 1: Overview on the ASTRA model



Source: TRIAS D3 report (Krail et al. 2007)

- Welfare measurement module (WEM) provides aggregate indicators like transport intensity, investment multipliers or cost-benefit ratios of policies.
- Infrastructure module (INF) calculates travel speeds depending on transport demand, taking infrastructure investments into account.

The strength of the ASTRA model is that the nine modules are not simply connected in a linear way, e.g. the economy driving transport and this leading to emissions, but that various feedbacks are implemented between the modules, such that inventions in the vehicle fleet (e.g. hydrogen cars) or new energy supply systems (e.g. renewables) feed back into the economic system through changes of investments or cost changes.

3 Merger functionality & module definition

The functionality of *Merger* and the rules for the design of the modules are closely connected to each other and result both from the needs defined by the task which has to be accomplished. Therefore, both are described synchronously.

Our intention was to transform ASTRA, our large, highly sophisticated SD model implemented in one single model file into a set of models for easier maintenance and development. We wanted each of these modules to be a valid SD model able to run alone. And, most important, this set of modules should still have the full functionality of the original single model, i.e. the ability to compute integrated results with all the feedback loops as described in the previous chapter.

First, we need to clearly separate between the two expressions model and module. Therefore, it is essential to distinguish between two perspectives: a purely technical one and the model logic. Taking the model logic perspective, a model consists of various modules (compare Figure 1), meaning that a module is a part of the model and one model consists of various modules. From a technical perspective a model is one single file. So a module can be a model, technically, if it is a standalone file. Therefore, we developed a set of conventions to be able to spit our large single file model into a set of modules implemented as standalone model files. And we developed a java-based tool, the *Merger*, to put these modules back together into one model file. The conventions developed enable the definition of interfaces between the modules and also provide a way to have data from other modules available in a module needing this data when run in standalone mode.

All developers implement any changes to the ASTRA model only in the modules. For running the complete model, *Merger* produces one single model, which then can be run.

So each module is designed to fulfil two functions:

1. Each module has to be a valid standalone SD model.
2. Merging any number of modules into one model has to result in a valid SD model which can be run without further manipulation.

Therefore, each module has to comply with the following structure and syntax rules. And following our formatting recommendations makes both the modules as well as the resulting model more readable.

3.1 Interfaces between modules

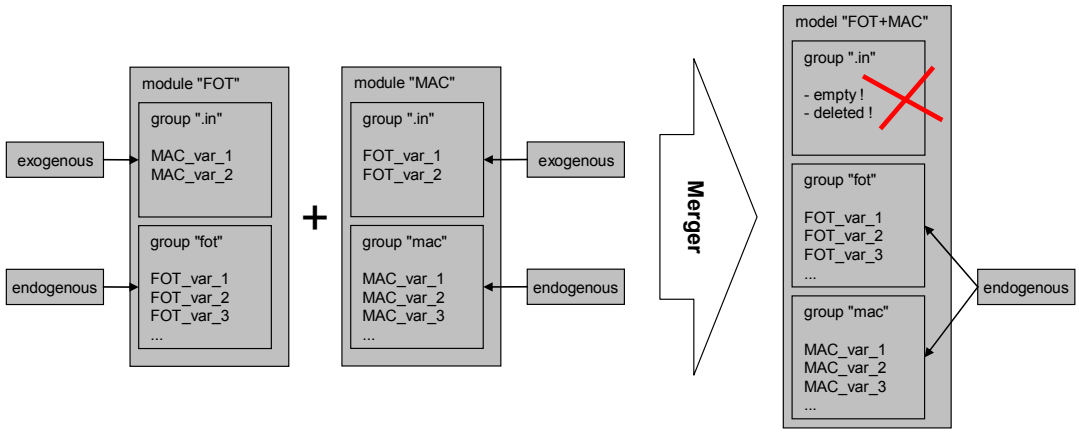
All modules are linked to each other by using variables from other modules. For splitting the model into modules, these interfaces had to be made explicit. Doing this, we could largely build on the work of Thompson and Bush (Thompson, Bush 2006).

Assuming the variable *FOT_imports* calculated in the module FOT (foreign trade) is used in the macroeconomic module (MAC). In the single model file version of ASTRA, this would mean that *FOT_imports* is simply used as an input variable to some variable definition in the MAC module, e.g. for *MAC_gross_domoestic_product*.

But in the standalone version of the MAC module, *FOT_imports* is still used in this calculation, but its definition would be missing. Therefore, we introduced it as a data variable, receiving its values from a vdf file (Vensim data file), making it exogenous to the standalone version of the MAC module. Due to this step, the MAC module can be run as a standalone model.

When *Merger* generates one single model file from the two modules, it deletes the data variable version of FOT_imports, and only the endogenously calculated *FOT_imports* remains, providing the MAC module with dynamic results of this variable. This is visualized in an example in Figure 2.

Figure 2: Example of the basic functionality of the *Merger*



Source: own diagram

For easier identification of the data counterpart of each interface variable, each module has to implement a group named *in* containing all the exogenous data variables calculated in other modules. Variables in this group have to be defined using the same subscript ranges as used in the definition of that variable. E.g., if a variable is defined with two equations for two subscript ranges, then the definition in the *in* group must use exactly this combination and not a single definition using a combined subscript range. This is necessary because only in that way merger is able to identify pairs of variable definitions forming the interface.

The example given in this chapter treats the least complex case of merging two modules. The design of the *Merger* is more general, and it allows the merging of any number of modules.

3.2 Special variable groups

There are variables which are defined in more than one module but cannot or should not be interface (i.e. data-) variables. E.g. lookup variables are of that kind or data variables which are exogenous to the complete model. These variables have to be placed in one of

the following special groups. In each module these variables have to be put in the same special group.

These groups are:

- Control
- Global
- Subscripts
- Data
- Venapp
- Policy

These groups are treated in a special way when the according modules are merged. All groups with the same name are merged into one and double defined variables are taken only once. This means that we can have e.g. in the data group exactly the variables needed by the according module. If another module also uses the same variable, then in the merged module the entry appears only once.

It is possible to use the same name for groups in different modules. Vensim makes one group of these by "reform and clean" the merged model. Therefore, any other group name besides the above listed can be used in various modules. But if the same variable is defined more than once, the ASTRA merger does not recognize this and does not remove the surplus definition, which results in an invalid model.

As mentioned above, *lookup* variables which are used in more than one module have to be put in the *Global* group. This guarantees to have the according variable only once in the merged model.

Additionally, the group *out* is used in every module to identify the variables which are used by other modules. This is needed for a better overview of the structure of the model which helps whenever the model is to be changed. The *out* group is not treated in any special way by *Merger*, it only serves for better structuring the modules.

3.3 Syntax of a module

The *Merger* breaks apart each module into its elements, analyse them and puts them together into one valid SD model file. In order to be able to do that, each module has to comply with essential syntax restrictions. Otherwise, Merger cannot fulfil its task error free.

(1) Underscores (or underbars as called by Vensim in dialog Tools/Options/Settings – Show underbars) have to be activated so variable names do not contain whitespaces.

(2) No space between variable name and subscript range, i.e. VFT_Car_Fleet_per_EU_country[EUCoun]

(3) No colon : in variable names

(4) No dot . in variable names

(5) No <-> sequence in variable names

3.4 Formatting of a module

The formatting rules are rather recommendations, not necessary restrictions. Following them makes it easier to identify the interfaces and make the views in the merged model more readable.

(1) Formatting of *in-group* variables: grey and italic

(2) Also, *in-group* variables should be shadow variables on the view level. If they are not, in the merged model all input variables will appear unformatted next to the variable in an unreadable way. This is hardly readable.

4 Merger technology

Merger was developed as a standalone Java application, build against Java 1.5. Its core functionality is built on Regular expressions, using the Java implementations included in the package *java.util.regex* described in the java documentation (Sun Microsystems Inc. 2004).

"Regular expressions" is a technology for finding text patterns, which is essentially what a computer is doing when interpreting computer program code. Therefore, they are very well suited for analyzing Vensim SD code on the ASCII text level.

Merger works only with the text version of Vensim SD models. It decomposes a set of given SD models using Regular expressions and recombines all elements into one valid SD model, omitting double defined variables.

The fundamental algorithm for combining various modules into one model is extremely simple:

For each variable in all *in-groups* of the modules, merger checks if this variable is defined in any of the provided set of modules. If this is the case, the variable definition in the *in-group* (by definition, a data variable, see above) is deleted and only the definition of the variable containing the calculation is kept, thereby the module which uses the according variable as exogenous data in its *in-group* when run as standalone model, now uses the endogenous calculation of that variable as it is merged with the module where this variable is calculated. Figure 2 gives an example of this. Also, if a variable from the *in-groups* is not found in the given modules, it remains as data variable in the *in-group* of the newly created model, remaining exogenous. This can be the case if only a subset of all modules is merged.

5 Version-controlled repository for ASTRA

Additionally to the modularization, we introduced a so called version controlled repository for storing the ASTRA model. This is a technology used for team-working in software development. We decided to use Subversion (available at subversion.tigris.org), an open-source software being a wide-spread standard in the software developing community.

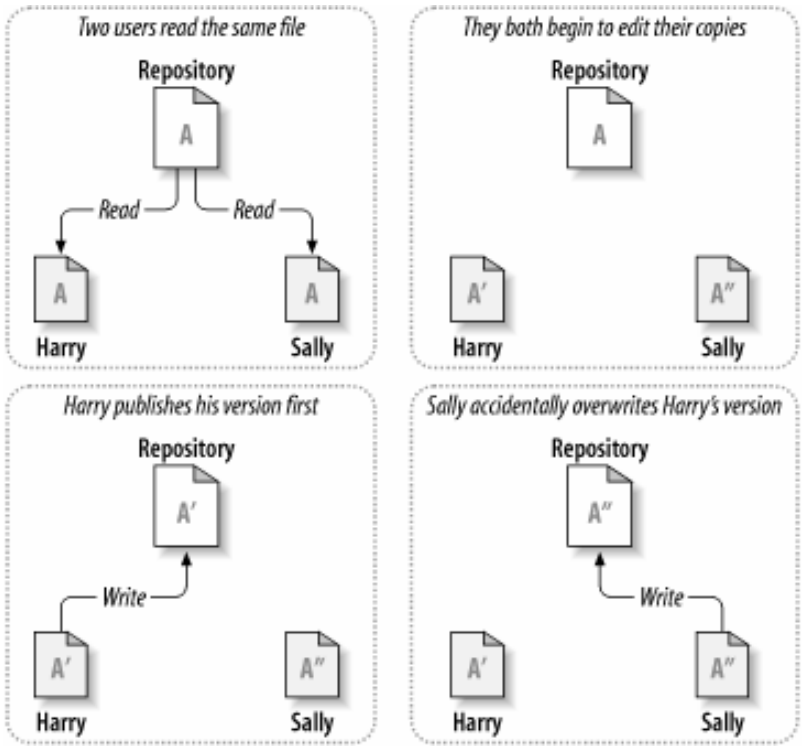
This technology offers various benefits:

- enabling a group of people to simultaneously work on a shared set of files,
- tracking changes on a set of files

- providing access to all previous versions of all files
- providing a reference so everybody has the possibility to always work on the latest version of these files.

This is achieved by simply always storing everyone's work in the repository and by always downloading the latest version from the repository before starting to work. The advantage compared to a normal centralized file server is that subversion notices and informs the user if he tries to overwrite a file which was modified in the meantime by a different user, which is the classical problem in a teamwork environment, visualized in Figure 3. In this case, the user is then able to merge the two different versions into one and make sure that the new version of the file still works.

Figure 3: The problem to avoid during distributed development



Source: Subversion documentation (Collins-Sussman et al.)

This is where the modularization of ASTRA turns out as highly useful. Since ASTRA has been split into more than 40 modules these kinds of conflicts are minimized.

Also, an essential feature of a version controlled repository is the access to all previous versions of all files. Therefore, it is only a minor problem if some developer publishes a not functioning module or if any other file turns out to be incompatible with the rest of the model files. Simply stepping back in revision history solves the problem.

The workflow using a repository is only slightly different compared to working the traditional way without version control. Using a repository, working with the relevant model files does not change. All model files are still located locally in a random folder on the modeller's hard drive. The difference is in having a common reference of the model files available via internet. And the local folder is connected to the shared repository.

Crucial is the fact that this connection is not automated in either direction. This means that the user needs to take an active decision on synchronizing his local work with the shared repository. This synchronization process has to be done separately for sending changes to the repository and for receiving changes from the repository. This is very convenient in every day work, due to many situations in which the user would like to get the latest changes in the repository, but does not yet want his work to be published or vice versa.

The resulting workflow is:

1. Start working by making a so called *update* ,i.e. copy the latest changes from the repository to the local folder.
2. Modify the local copy, called the working copy.
3. Whenever achieving a stable version, *commit* the local changes to the repository, i.e. send the local modifications to the repository and thereby publish these changes within the workgroup.

The security level of our repository is sufficient for our needs. Subversion is plugged into an Apache server, the connection is established via an https connection, with every user having a user name and password. There are probably ways to hack into our system, but it is most definitely more secure than sending our model files via non-encrypted emails, which was the standard before using our repository.

6 Practical workflow using Merger

Eventually, we would like to shortly present the steps we had to take to split ASTRA into more than 40 modules and the steps we take whenever we want to generate a complete model from our modules.

6.1 Splitting up ASTRA

Splitting up ASTRA was by far the most work intensive task of the whole project. We worked in several steps. First, one developer had to split the complete model into three parts, introducing the interfaces and conventions as described above. Then, each part was assigned to one partner institute. There, the splitting continued. During an intermediate status we had all nine functional modules available as standalone models. And finally, these were further split to eventually more than 40 modules.

In order to produce the exogenous data necessary for the interface variables of each module, a *savelist*[1] was compiled for each module containing the variables of the *ingroup*. With this *savelist*, we export exactly that data from a complete model result set (a so called *run*) which we need as exogenous data for the standalone module. The compilation of this savelist is also automated using regular expressions.

[1] Vensim terminology for a list of variables used to extract data from a model run result set.

6.2 Merging modules into one model

Producing one valid SD model from all our modules or even from only a subset thereof is nothing more but executing a single batch file which starts the *Merger*. All possible settings are stored in a configuration text file and a text file containing the list of all modules that should be included in the merge. The configuration file is used for defining the file list of exogenous data files of the resulting merged model as well as the name for the next run with that new model.

These configuration possibilities enable us to execute the *Merger*, open the newly generated model with Vensim and start the simulation directly, without any further modifications needed. This makes it a highly practical tool for our needs.

7 Conclusion

We would like to stress the highly practical every day use the described transformations brought to us. We experience a significant facilitation of the maintenance of our highly sophisticated ASTRA model due to the introduction of the modular design as well as the appliance of a version controlled repository. It was a necessary step, and we would take it again if we had to make the same decision with the experiences we have today. We are convinced that this is a very good way to develop and maintain such a complex model. Without modularization and a version controlled repository, significantly more time would be needed for purely repetitive tasks not related with the content of our work. The implementation of the described steps gives us notably more time to do quantitative policy analysis rather than purely technical maintenance as it used to be before we completed this project.

Naturally, there are problems with our new way of working which should not be omitted. When working with a version controlled repository, it is possible that a developer publishes a module which is not compatible with the other modules anymore. If he does so, he did not test with the rest of the modules in the repository before. This is a clear violation of the rules of team development, but it happens, simply because it is possible. This case is not a major problem, because any team member has access to all previous versions at any time, but it causes trouble.

Nevertheless, this example of trouble according to our experience is very insignificant compared to the major advantages inherent to a version controlled repository and a modularized large scale SD model.

List of figures

References

Collins-Sussman, B.; Fitzpatrick, B.W.; Pilato, C.M. Version Control with Subversion - For Subversion 1.4. Online: http://svnbook.red-bean.com/.

Krail, M.; Schade, W.; Fiorello, D.; Fermi, F.; Martino, A.; Christidis, P.; Schade, B.; Purwanto, J.; Helfrich, N.; Scholz, A.; Kraft, M. (2007): Outlook for Global Transport and Energy Demand, Deliverable 3 of TRIAS (Sustainability Impact Assessment of Strategies Integrating Transport, Technology and Energy Scenarios). Funded by European Commission 6th RTD Programme. Karlsruhe, Germany.

Schade, W. (2005): Strategic Sustainability Analysis: Concept and application for the assessment of European Transport Policy, Dissertation Thesis (2004) at University of Karlsruhe, NOMOS-Verlag, Baden-Baden.

Schade, W.; Helfrich, N.; Wietschel, M.; Krail, M.; Scholz, A.; Kraft, M.; Fiorello, D.; Fermi, F.; Martino, A.; Schade, B.; Purwanto, J.; Wiesenthal, T.; Christidis, P. (2008): Alternative Pathways for Transport, Technology and Energy to promote sus-tainability in the EU, Deliverable 4 of TRIAS (Sustainability Impact Assessment of Strategies Integrating Transport, Technology and Energy Scenarios). Funded by European Commission 6th RTD Programme. Karlsruhe, Germany.

Schade, W.; Krail, M. (2006): Modeling and calibration of large scale system dynamics models: the case of the ASTRA model, paper presented at the 24th International Conference of the System Dynamics Society in Nijmegen, The Netherlands, July 23-27th 2006.

Sun Microsystems Inc. (2004): Java 2 Platform Standard Edition 5.0 Documentation. Online: http://java.sun.com/j2se/1.5.0/docs/api/.

Thompson, D.; Bush, B.W. (2006): Group Development Software for Vensim, paper presented at the 24th International Conference of the System Dynamics Society in Nijmegen, The Netherlands, July 23-27th 2006.