Towards a Multi-Purpose Framework for Tax-Benefit Microsimulation

Cathal O’Donoghue
(University of Cambridge)

Data: 05/04/2001 (quinta-feira)
Horário: 16h

Realização:

Local:
Praia de Botafogo, 190 - 10º andar
Auditório Eugênio Gudin

Coordenação:
Prof. Pedro Cavalcanti G. Ferreira
e-mail: ferreira@fgv.br
Towards a Multi-Purpose Framework for Tax-Benefit Microsimulation
A Discussion by Reference to EUROMOD, a European Tax-Benefit Model1

Herwig Immervoll and Cathal O'Donoghue.

08/03/00 Version.
Draft. Please do not quote.

1 Introduction

The purpose of tax-benefit microsimulation models (MSMs) is to provide a tool for policy analyses enabling researchers and analysts to concentrate on formulating policy instruments and analysing results while relying on the model to apply the policy instruments to the microdata. From the user's point of view, numerous characteristics and features are desirable for the tax-benefit model to be useful. This paper will consider some of these design issues and illustrate possible approaches by reference to the EUROMOD tax-benefit model.2

Tax-benefit models save researchers and policy analysts the considerable time and effort involved in data handling and model construction. If, however, the structure of the model were too rigid to simulate a certain policy of interest, it would have to be reprogrammed to a certain extent, which partially defeats its original purpose. One challenge lies in constructing a tax-benefit model flexible enough to allow the simulation of far-reaching and \textit{ex ante} unknown policy alternatives while keeping the specification of relatively minor policy changes reasonably simple.

2 Objectives and Desirable Features of Tax-Benefit MSMs

This section outlines objectives and desirable features of microsimulation models used for tax-benefit analysis. Previous studies which have focused on the overall design of tax-benefit models such as (Krupp, 1986; Hoschka, 1986; Caldwell, 1986; Sutherland, 1991; Merz, 1991; Sutherland, 1995; Harding 1995; Citro and. Hanushek, 1991a, 1991b) have been largely broad based in approach or focused more on data issues. [?? Add literature description here, get references for Wolfson's work] Our paper focuses on detailed computational design of static microsimulation tax-benefit models.

In order to set the scene as to how a microsimulation framework should be created for tax-benefit modelling, we first discuss the potential demands placed on a tax-benefit model.

1 Immervoll and O'Donoghue are both Research Associates at the Microsimulation Unit in the Department of Applied Economics at the University of Cambridge. This paper was written as part of the EUROMOD project, financed by Targeted Socio-Economic Research programme of the European Commission (CT97-3060). We are grateful to Holly Sutherland and participants at the EUROMOD advisory group meeting (Feb. 11, 1999, Copenhagen) for helpful comments. The authors alone are responsible for any errors, as well as the views presented in this paper.

2 Further information on the EUROMOD project is provided in Immervoll et al (1999).
Static microsimulation tax-benefit models are used by academics and policy analysts within government and elsewhere to study the effects of tax-benefit policy as well as for designing new policy instruments. Designers and Analysts of tax-benefit instruments often have different objectives. Sutherland (1991) has outlined a number of possible objectives:

- Revenue neutrality
- Redistribution of cash towards families with children
- Improvement of work incentives
- Lowering of income tax rates
- Producing a tax system that is accepted as "fair"
- Targeting expenditure particularly on the disabled and those caring for them
- Simplifying the tax and benefit systems and reducing administrative costs
- Reducing dependency on means-tested benefits.

It is the role of tax-benefit models to assist in the analysis on under these headings.

The objectives outlined above are broad policy requirements. We now outline some practical requirements and considerations:

- Models need to be able to cope with regularly changing tax-benefit systems. In many countries, systems change from year to year.
- In addition to tax-benefit systems changing regularly, the datasets that are used as inputs into the model also frequently change and are uprated. Therefore model frameworks need to be able to handle new data with ease.
- Teams using these models can be large and thus the model may be required to be used by non-programmers.
- Analyses may have to be done in a short period of time – this may especially be true in a government environment.
- Another possible requirement is the desire to compare the effectiveness of tax-benefit instruments in different countries. Research has shown that simply lining up national models next to each other is not suitable (Callan and Sutherland 1997). This is because of the numerous possible decisions one has to make when creating a model. In different national models different decisions are often made in the light of national priorities.

How do we design a model framework which meets so many diverse objectives? Examining the objectives more closely, these requirements can only be met if the model is generalised, transparent, modular and well documented.

The first decision, which needs to be made, is to decide how much generalisation should be used. The degree of generalisation relates to the level at which a model is parameterised so that model code can be used for different purposes without recoding. While generalising as much as possible makes the model more flexible, it also has the effect of making it more costly and difficult to develop and also potentially making it less transparent than a similar less generalised model. The decision about how much generalisation should be used very much depends upon the objectives of the model. If the model is for a narrow specific use with a planned limited life then, it makes more sense to produce a less generalised model. However in order to meet the flexibility required above to be able to handle different tax-benefit systems and datasets, whether this be across time or across countries, then a high degree of generalisation is important. This allows comparability to be maintained. Designing
model frameworks are very demanding in terms of resources. In a survey by Mot (1992), it was generally found that static national models generally took 2 to 3 man-years to produce the model. More sophisticated models such as the TRIM2 model in the USA took much longer. Although more costly to build the initial framework, a generalised model is less costly in the long run as code can be reused and thus allow for major economies of scale.

Elements of the model which can be generalised include:

a) the data handling component,
b) the output component,
c) the data structure declarations and memory management component,
d) the model "manager" which manages the order of the tax-benefit routines,
e) the tax-benefit routines.

These headings can be divided into components which deal with operation of the program (a-d) and those which deal with the calculation of the tax-benefit routines (d). It may be possible to see the former as a black box which users rarely need to see and the latter as a glass box which is of most interest to users. Generalising the "black box" component is the most logical as in this way the framework can be used for different uses without analysts having to recode it. Generalising the "glass box" is less straightforward. As this is the part of the code which is likely to be more used by analysts, it must therefore be more transparent to them than the components which they do not need to see. For example it may be possible to design a tax allowance routine which is general enough to handle very many different types of tax allowance, but in the process may become so complicated as to be rendered unusable by the users of the model. Therefore a library of specific routines rather than a generalised one may be more appropriate. However on the other hand, it may be quite satisfactory to design a generalise a routine which quite often uses the same format such as a tax schedule. One therefore cannot make a definitive decision as to the level of generalisation in the tax-benefit routines" but judge each component on merits.

From a practical point of view elements of the tax-benefit component which should be generalised include:

a) the set of variables to be used,
b) the definitions of aggregate incomes,
c) the definitions of the units of analysis,
d) flexibility in the definition of sharing rules
e) the definition of model components
f) the order in which sub-components of a tax-benefit system are run,

If the set of variables can be generalised, then it makes it easier to add new data as no new coding is necessary when new data are added. By generalisation here we mean that we do not declare large series of variables within the model, create memory space for them separately and initialise them elsewhere. Instead if there is a general framework to do this, we can do this automatically for externally specified lists of variables. Thus the framework effectively starts with no variables and only used variables, which have been specified. In this way different sets of variables as in the case of different national datasets can be included without reprogramming. In addition, generalising in this instance improves transparency, as there is less sources of error when the inclusion of a new variable requires the variable to be included in one place rather than many.
Another source of generalisation is in the definition of aggregate incomes. Within a tax-benefit system aggregate variables are often required. This may include such things as the taxbase for estimating the amount of tax liabilities, the amount of an aggregate income for classifying whether an individual is considered dependent or in the outputs of the model such as disposable income. There can be a significant number of these in a model and also the definitions of components can vary between uses as in the case of the definition of the taxbase in different tax systems. Without generalising the determination of aggregate incomes, changes in definition would require the model to be recoded for each use, again a potential source of error. Thus generalisation in this case allows improves flexibility, improves transparency and improves the robustness of the model.

In addition to aggregate incomes, another issue is the flexibility of the analysis unit to be used. Frequently data is provided with rigidly designed units of analysis such as individual or household and occasionally also as nuclear families. However they do not always agree with the definition of a family used for tax-benefit purposes within a country. Indeed frequently the definition of a family unit can differ between instruments within a country. In the case of a once off model, it makes sense to program one set of units, however if one wants to use the model framework for different purposes then one should generalise the definition.

On a related issue given the issue of different units of analysis, one also needs to think about the sharing of resources within a unit. If one ignores this issue then results can only be considered at the widest unit of analysis of a tax benefit system. For example if housing benefits are paid on household housing costs, then if one wants to look at the distributional impact of a tax reform on nuclear families, then how does one deal with housing benefits? Are the given to the head of the household or split between the members? If one splits income then one needs to define sharing rules.

Another less obvious source of generalisation is in the actual components of the tax-benefit system. Because of large amounts of data, complicated algorithms and large numbers of parameters and tax-benefit instruments, tax benefit MSMs are inherently complex. In designing such systems, it is essential to break the overall complexity down into manageable modules. Each module should have a clearly defined purpose. Each module should constitute a “black box” with clearly defined boundaries and inputs/outputs. Any “side effects” on other modules are to be avoided. Applying this design philosophy ensures that individual modules can be developed independently. Once a module has been thoroughly tested and is found to work, it can be added to the system as a whole. This is a method also employed in varying degrees by the US model TRIM2 (Mot, 1992) and the Institute for Fiscal Studies Model, TAXBEN (???, 1997). Equally important, a strictly modular design simplifies the maintenance of a system considerably. It ensures that changes, which inevitably have to be made during the lifetime of any tax-benefit model, do not have unexpected side effects on other modules.

However not only is the model modularised, but also the modules themselves can be generalised. Thus each module is a derivative of a basic template with the same input and outputs data structures, with only the core algorithm changing. Except for the core algorithm, modules then only vary by changing parameters (described below). In generalising the structure of the building blocks, we have the advantage that the code required run a particular

---

module can be simplified. Furthermore it allows the order in which modules are run to be
generalised. Although not an important issue in the construction of a national tax-benefit
model, for a model framework to be applied to different countries this is a strong
requirement.

In a generalised model, the essential components need to be kept separate both logically and
physically. "Logical" separation means that the modules, which can be modified by the user,
have limited access to the model’s variables and parameters. For example, constants and data
related to the model’s internal structure cannot be changed from within the tax-benefit
modules. In addition, functions are provided for accessing frequently used characteristics of
tax/benefit units (e.g., NumberChildrenInTaxunit, IsMarried, IsLoneParent, etc.) so that users
don’t have to access these variables directly. This approach ensures a consistent
interpretation of variable values (especially categorical variables, such as marital status)
across all modules and considerably simplifies the maintenance of the model.

"Physical" separation means that the components of one module are stored together rather
than scattered across different files, spreadsheets, etc. For example, all the operations
necessary for calculating a certain income tax deduction (determination of tax unit,
aggregating income data across the tax unit, accessing parameters as stored in the parameter
lists, performing the actual calculation of the deduction) will be contained in one single
function. This function will be stored separately from other components of the model. This
modular design helps preventing errors during the initial model construction. It also makes
debugging and model validation more efficient and simplifies the maintenance and actual use
of the model. Most important for the user, it simplifies the process of making changes to the
structure of tax-benefit rules.

The use of modularisation therefore meets a number of objectives outlined above. The use of
modules which can operate independently means that new components can be added quickly
and without distorting the operation of the rest of the model. Without this type of structure,
adding new components, because of the complexity of the program can have many
unforeseen consequences and thus increases the possibility of error. Being able to add or
eliminate modules at will allows one to adapt the model to changes in tax-benefit rules
relatively easily and as a result it can help users respond quickly to policy questions.

Another advantage of modularisation is that it helps to reduce the complexity of a system by
focusing on simpler building blocks. This not only helps the programmers in designing code
for a system, but also allows non-programmer analysts to follow the operations of particular
parts of a tax-benefit system without being exposed to the complexities of the whole model.

One further desirable objective is robustness. The tax-benefit systems which tax-benefit
models are designed to analyse are highly complex and thus the models themselves by
definition are complex. As a result the possibility of error increases. Furthermore attempts to
generalise the model framework also add to this as users have many possibilities to change
the operation of the model. Therefore it is important to ensure as much as possible that what
users try to do with the model are correct. For example, if the order in which modules are run
can be varied and if module A depends on module B, then the model needs to prevent module
A being run before module A. In addition, if the input datasets can be varied then the model
needs to ensure that the correct set of variables required for a particular system have been
read in.
3 The Structure of Tax-Benefit Systems

A general tax-benefit modelling framework will ideally be able to accommodate any existing or hypothetical tax-benefit system. In designing such a framework it is therefore essential to identify the principal elements of tax-benefit systems. In other words, it is necessary to find a suitable "common denominator" of all (reasonably) possible structures. However, in general, there exists a trade-off between structure and flexibility: The modelling framework needs to provide the structure necessary for setting up a simulation model without limiting the breadth of tax-benefit systems that can be simulated. In "real world" tax-benefit systems, elementary policy rules are grouped together to form identifiable blocks such as "instruments", "policies", etc. In modelling a country's system, it is desirable to match the "real" system's hierarchy as closely at possible so that the logical representation provides a good intuitive equivalent of the original. Figure 1 shows the hierarchical structure that we propose for a general tax-benefit modelling framework. The actual tax-benefit algorithms are structured as follows. Each tax-benefit system is made up of individual policies. These are elementary collections of tax-benefit instruments. Examples for a policy are Income Tax, Social Insurance Contributions or Social Assistance Benefits. The policy spine is a list of policies indicating the sequence by which they are applied in the tax-benefit system, i.e., if social insurance contributions are tax deductible, then the entry Social Insurance Contributions would have to appear before Income Tax because the model needs the amount of social insurance contributions as a prerequisite to calculating income tax; similarly, if social assistance benefits depend on after tax income, then the entry Social Assistance Benefits would have to appear after Income Tax since income tax is a necessary input for calculating social assistance benefits. At the lowest level is the tax-benefit module, which performs the calculation of a certain part of the tax or benefit (e.g., a deduction, or applying a rate schedule to a tax base) on each fiscal unit. The modules represent the elementary building blocks of the tax-benefit system: Only the modules contain actual tax-benefit rules. All other levels are merely necessary to structure these rules and apply them in the correct sequence.

4 In some cases, different policies may be optional in the sense that the tax payer/benefit recipient can choose from different alternatives (e.g., different alternative benefits). In such cases, it is necessary to simulate all the individual options and then apply some decision rule for choosing between them (e.g., by assuming a decision which would maximise disposable income). The proposed framework allows for this by providing such decision-rule elements. These can be inserted anywhere along the policy "spine".
4 Key Concepts for Tax-Benefit Modelling

As discussed in section 2, modelling flexibility is to be achieved by enabling the user to make as many adaptations and definitions as possible without changing the program code itself. This is usually done by allowing the user to make changes to certain sets of parameters. The "parameterisation" of calculations ensures that their operation remains transparent and adaptable. Parameterisation also facilitates experiments with different parameter values. For example, one could wish to analyse how including unemployment benefits in the income tax base, changing the definition of the tax unit (e.g., changing the maximum age of a "child"), or altering the number of tax brackets affect the distribution of disposable income. All such changes can be specified by setting appropriate parameters in the parameter lists while no modifications need to be made to the model's program code. The actual tax-benefit algorithms are coded as functions of these externally defined parameters. If the modelling framework is to be kept general enough to accommodate many different tax-benefit systems, then a very large number of parameters are needed to specify a tax-benefit system. To keep manageable and transparent, it is desirable to organise parameters according to their purpose. In the framework proposed here, policy parameters relate to one of the three levels of the hierarchical structure discussed above:

1) "policy spine": This set of parameters is simply a vector of policy names. It specifies the sequence of policies.

2) "policy": This set of parameters is represented by a vector which contains the module names in the desired order within the policy.
(3) "module": Parameters at this level enter the actual tax-benefit algorithm contained in the module.

Which parameters are needed at the module level depends on the type of calculation performed by the module. Here, the following categories of parameters can be distinguished.

4.1 Definition of Fiscal Unit
Tax-benefit rules relate to certain fiscal units. Each module must contain the name of the type of fiscal unit on which the tax-benefit algorithm is to be performed. Fiscal unit types themselves can be defined in a separate parameter sheet. In the simplest case, the fiscal unit type is either the entire household or the individual. If not then one has to define which members of the household belong to the same fiscal unit. At present, possible choices are Cohabiting Partner, Married Partner, Child and Dependent Parent. All these choices relate to the "head" of the fiscal unit. For the latter two, a powerful set of conditions is available for defining what constitutes a "child" or a "dependent parent" (e.g., age limits, income limits, conditions relating to marital-, labour market-, or education status, etc.). All these conditions can be combined with logical AND and OR operators. Appendix 1 of the paper explains the method which the framework uses for assigning individual members of a household to the different fiscal units.

4.2 Sharing of Benefits and Tax Burdens within the Fiscal Unit
For assigning the tax or benefit amount that is computed by the tax-benefit algorithm in the module to the appropriate person(s) in the fiscal unit, it is necessary to provide information about sharing arrangements. The framework supports a number of different arrangements (e.g., equal sharing among members/parents/adults/children of the fiscal unit; "head of fiscal unit" gets is assigned everything; mother/father gets everything, etc.). By allowing such explicit definitions of intra-unit assignments of taxes/benefits, it becomes possible to analyse simulation results at any level of aggregation.

4.3 Instrument Specific Parameters
These are parameters such as amounts, limits, rates, schedules, eligibility conditions, etc.

4.4 Income Concepts
Income concepts used in the tax-benefit algorithms (e.g., taxable income, "means", etc.) or as output of the model (e.g., disposable income) can be defined in terms of all monetary variables (whether contained in the micro-data or simulated by the tax-benefit model) available in the model. Each income concept is defined in terms of a vector of numbers between −1 and +1. The size of the vector is equal to the number of monetary variables in the model. For each of the variables, the numbers in the vector indicates what fraction this monetary variable is part of the income concept. For example, if "mortgage interest payments" are deductible from taxable income then the "taxable income" vector would contain a "−1" for the "mortgage interest payments" variable.

Figure 2 illustrates typical module parameter sheets. Each module starts with the name of the module (co_it_main_tfa; co_it_schedule) followed by a number of parameters. Parameters common to each module are "TAX_UNIT" and "SWITCH". The former of the two
specifies the fiscal unit relevant for this module while the latter determines whether or not this module should be included in the calculations. All other parameters are module-specific. The sequence of the module determines the sequence of calculations in the simulation run. For instance, in the case shown in figure 2, the main tax free allowance is computed before the income tax schedule is applied to the TaxBase (the reason, of course, being that the tax free allowance is subtracted from the tax base). This design permits re-arranging the order of modules in any order necessary without having to alter the tax-benefit program code. The sequence can be changed by simply moving around the parameter blocks in the parameter sheet.

This concept of modules as distinct building-blocks of the simulation has special advantages in a multi-country context. By using the same model framework for different tax-benefit systems, one is building up a large library of existing modules. Modules that have been designed for one particular purpose can be re-used in other contexts so that when it is necessary to incorporate a new tax or benefit instrument, it will often not be necessary to program any new tax-benefit rules. The flexible order of modules and the high level of parameterisation ensures that the same modules can be used for a multitude of different purposes. In addition to modules that have been designed for a specific purpose in a specific country, the EUROMOD framework also provides a large number of “general” modules which were designed without any single country or specific purpose in mind. These “general” modules provide a high degree of flexibility and can be used for many different purposes. Examples are schedules where the number of rates, etc. is flexible and where the income base to which the schedule is to be applied can be freely defined (co_it_schedule in figure 2 above). There also exist a number of standard deductions, allowances, etc. One of the most powerful set of “general” modules is available for implementing benefit rules. General modules exist for defining eligibility conditions, means and disregards for means-tested benefits, and equivalence scales for determining the benefit amount as a function of characteristics of the fiscal unit (such as age, number of people in the family, number of...
children, etc.). These modules currently provide more than 400 different parameters, which makes them extremely flexible. All these parameters can be specified in parameter sheets which means that in many cases, even very complicated benefits can be implemented without any need for programming. Apart from the considerable amount of time and effort that can be saved by re-using already existing building blocks, there is the added advantage that these general modules have already been thoroughly tested. One can therefore be confident that the risk of programming errors is minimal. This means that the user can focus his or her efforts on specifying policy rules and analysing different policy scenarios rather than having to worry about programming technicalities.

In addition to policy specific parameters, a number of parameters relate to the micro-data on which the tax-benefit system is to be simulated. All variables used in the model are specified in a list containing the variable names and additional information such as whether the variable is an individual variable or relates to the household as a whole; whether or not it is a monetary variable, etc. In addition, uprating factors can be specified for each (monetary) variable in order to permit up-rating the micro-data from the data year to the policy year of interest. For cases where certain variables are not available in the micro-data one wants to use for the simulation, default values can be specified. This is especially important in a multi-country context, where one may want to simulate the effects of introducing a tax-benefit instrument from country A in country B. If the tax-benefit rules of this instrument require a variable which is not available in country B micro-data then one can specify appropriate default-values for this variable. Default values can be specified either directly (i.e., by specifying the actual value) or by referring to a variable that is available in the micro-data and is considered a good approximation of the missing variable. For example, in a situation where the tax-benefit rules of a country A instrument require information on whether someone is a "civil servant" and where there is no "civil servant" variable in the country B micro-data, one can specify that the variable "public sector" should be used as a proxy for "civil servant".

5 Program Architecture of the Framework

Fig. 3 shows the basic elements of the program and how they relate to each other. After the parameter files, which contain the information necessary to run the tax-benefit algorithms, have been read the model reads one household at a time. Based on the tax unit definitions read in previously, the program then decides which individual belongs to which fiscal unit. Once all the fiscal units which are used in the tax-benefit system have been formed, the household is passed through all relevant rules of the tax-benefit system.

The last step of the program is writing the output of the simulation. The framework provides a standard output routine which covers most of the "popular" indicators and statistics. In addition (and depending on data access restrictions), it is also possible to generate a micro output file (i.e., all output variables for each household/individual/etc.). This micro output file can then be used with any statistical package for performing more elaborate analyses. A special feature of the output routine is that it can be integrated into the "spine" of the tax-benefit system just like any other policy. This means that it is possible to produce output at any level of the "spine". The user can, for example, specify that he or she wants output to be generated both before and after the Social Assistance policy instrument. By comparing different outputs, one can then easily observe the differential impact of one individual policy or a set of policies. Of course, the typical spot of the output routine will be at the very end of the policy spine in order to show the outputs for the tax-benefit system as a whole.
Building a tax-benefit MSM requires a very significant investment of time and resources. It is, therefore, important to prevent a situation where the burden of learning how to use the model is such that only the model-builders end up using it. Also, in order for a tax-benefit model to remain accessible and usable for a long time, it is necessary not to irrevocably bind it to one computing environment. In choosing the environment and programming language of the model, an effort has been made to ensure the longevity of EUROMOD (figure 4).

Originally, the model is designed for a Windows environment since this platform is familiar, easily available and comfortable to use. However, care has been taken to avoid a rigidity, which would prevent future adaptations to other platforms such as UNIX. The programming language used is C/C++. A language that is, and for a long time will be, available for all major operating systems. C/C++ facilitates efficiency in programming. However, the ability of C/C++ to write very streamlined and "direct" algorithms sometimes reduces the readability and transparency for less experienced users. As a rule, where trade-offs existed between transparency and speed, we accepted decreases in the model's speed in return for improved readability and usability.

By using a method for database access which is available for all major relational database management systems, database systems other than the one used as a default can be used for
data storage and management. All Windows specific modules of the program are kept separate from the rest of the model. When adapting the model for use on another platform, this allows the programmer to concentrate on the components which have to be changed (such as the user interface) while leaving platform independent components untouched.

EUROMOD System Environment

User-friendliness is frequently used synonymously with "Graphical User Interface" (GUI). Although GUIs can and in most cases do represent a major step towards easier and more natural to use software, most users of software running on GUI-based operating systems such as Windows would probably agree that not all such application programs are equally user-friendly. If the program is not based on a user-friendly design throughout, then adding a GUI will not save it from being hard to use. It is, therefore, important not to spend major programming efforts on GUIs without considering the underlying conceptual model design. Indeed, if the model design follows some basic principles, it can be made reasonably user-friendly even without a GUI (which can, of course, still be added at a later stage).

In designing EUROMOD, an attempt has been made to

(1) Utilise users' knowledge of standard software packages.
Standardised interfaces are used in order to allow the use of familiar standard software packages to view and manipulate the model's input/output data. All parameter lists are stored as spreadsheet tables. They can be read and manipulated with any spreadsheet software (Microsoft Excel is used as default). Both the input microdata and the model's micro-output (simulation results) can be stored in one of the widely used relational database systems (Oracle, Microsoft SQL, etc. MS Access is used as default). It is, thus, possible to utilise the powerful built-in features of the database management system to analyse the data. Input and output data are stored in two separate databases. This way, the input microdata can remain "read-only". However, the relational data structure makes it possible to combine the physically separate input and output data into one logical table to analyse the impact of a tax-benefit system in relation to all sorts of characteristics (age, household size, etc.).

(2) Use standard interfaces to facilitate the use of EUROMOD in connection with existing or future software

Given the limited resources of any microsimulation project, it is sensible to concentrate on the essentials rather than "reinventing the wheel". We recognise the desirability of a comprehensive user-interface that allows the user to perform all the operations involved in specifying, simulating and evaluating alternative policies in a single environment. For many reasons, however, it is important for a model to deliver results as soon as possible. Waiting for the completion of a professional user-interface, which can reasonably be expected to consume one fourth of the overall time needed to build the entire model, is a luxury most project teams cannot afford. It is, therefore, important to design the model's external interface in a way that (a) makes the model usable at an early stage; (b) reduces the need to re-develop features and operations which can be performed by existing standard software; (c) keeps the model open for future extensions. All of EUROMOD's parameter lists are stored in a standard spreadsheet format while the input and output databases are stored and accessed via a relational database management system. This enables the user to perform manipulation and evaluation of data in familiar and comfortable environments (see point 1 above) while initially keeping user-interface related programming efforts to a minimum. Furthermore, the transparent definition of the model's interface is a prerequisite for safe and smooth future extensions, such as developing a comprehensive GUI or embedding the model in a more complex dynamic microsimulation system. For these extensions, no changes will have to be made to the actual model. Any future extensions can regard the model as a "black box" to which they will "talk" by means of parameter lists and/or the input/output microdata stored in the relational database.

7 References


---

5 At a later stage, it is possible to allow for an interactive modification of parameters from within the model's user-interface.
6 Again, it should be noted that the model only permits micro-based output in cases where the data access restrictions do not prohibit this.
8 The model itself communicates with the database system through the standardised ODBC (Open Data Base Connectivity) interface. ODBC provides an interface that makes it possible to access different database systems with a common language. ODBC is available for Windows, OS/2, Apple Macintosh and UNIX platforms.


Appendix 1. Algorithm used for Assigning Individuals to Fiscal Units.

Tax unit types ("individual", "household", "family", "married couple", etc.) are defined in a parameter list in terms of conditions that need to be satisfied for a person to belong to a tax unit type. In each household, there exist one or more instances of each tax unit type. A specific instance of a tax unit type is a tax unit. Each tax unit has a number. Tax units of different types are numbered in parallel so that, for example, there is a tax unit no. 0 of type "married couple" and a different tax unit 0 of type "family". For each tax unit type, each person in the household receives a number indicating the tax unit (of this type) he/she belongs to. Everybody with the same number is in the same tax unit. Using the conditions mentioned above, it is possible to decide for each person, whether or not this person is member of a specific tax unit. A person can, of course, be member of more than one tax units simultaneously but he/she can only be member of one tax unit of a given tax unit type. A tax unit can be fully or partly occupied. For example if the tax unit type is "married couple" then in a one-person household, this one person is a tax unit of type "married couple" even though there is no spouse. Persons who are not assigned to a tax unit together with other persons form their own tax unit. For example, if the tax unit is "Married Couple without Children" and the household consists of a married couple and two children then there are three tax units of this type: tax unit no 0 contains the two spouses; tax unit no 1 contains the first child; and tax unit no 2 contains the second child.

Pseudo Code of Tax unit Calculations for a household HH:

For all tax unit types do the following:

{ IF tax unit type="household" THEN put all persons in HH into same tax unit number
  IF tax unit type="individual" THEN put all person in HH into different tax unit number
  IF tax unit type=something else THEN do CALC_FAMILY_TU
}
Routine to calculate specific tax units (CALC_FAMILY_TU):

For all tax units (maximum is number of persons in household) do the following:

{ For all persons in HH do the following:

//assign first person to tax unit:
IF (person is adult OR household has no adults) AND
person has not been assigned to a tax unit of the current tax unit type AND
no other person has been assigned to the current tax unit THEN
assign current person to current tax unit

//assign spouse of first person:
IF spouses are part of tax unit type AND
person is married to first person of tax unit THEN
assign person to tax unit

//assign cohabiting partner of first person:
IF cohabiting partners are part of tax unit type AND
person is cohabiting partner of first person of tax unit THEN
assign person to tax unit

//assign elderly dependents of first person:
IF elderly dependents are part of tax unit type AND
person is elderly dependent of first person of tax unit THEN
assign person to tax unit

//assign children of first person:
IF children are part of tax unit type AND
person is child AND
person is child of first person of tax unit THEN
assign person to tax unit

} }