Software Development of the Denver Activity-Based Model – Process and Results

Jennifer Malm, Denver Regional Council of Governments
Suzanne Childress, Denver Regional Council of Governments
Erik Sabina, Denver Regional Council of Governments
DeVon Farago, DeVon Farago LLC
Gerard Vaio, Cambridge Systematics, Inc.
Scott Meeks, Cambridge Systematics, Inc.

Introduction

This paper describes the design of the software developed by the Denver Regional Council of Governments (DRCOG) and its consultant team for DRCOG's activity-based model (ABM), referred to as the Integrated Regional Model (IRM), as well as the thinking behind key design decisions, and initial performance characteristics of the software. The first draft of the software is now complete, and the IRM is undergoing calibration/validation, working toward a completion target of March, 2010.

Motivation for Design Approach

DRCOG took an approach to software design that is in many respects strongly different from all the other ABMs of which we are aware. That decision was arrived at with some difficulty, but a long list of objectives for the system all pointed in the direction of an architecture making extensive use of information technology (IT) industry-standard tools and design. The objectives included the need that the system be:

- Explainable and understandable – this implied a high level of abstraction in design. For example, rather than trying to define choice sets for each discrete choice model, we designed software that used generic choice components that simply have "choices."
- Replicable and maintainable – the high level of abstraction in code design resulted in more generic code. Generic code is more re-usable by different model components, reducing both the development and maintenance effort.
- Scalable – future year scenarios are bigger in disaggregate models than are current year scenarios (more people, more jobs, more households, etc.), so the system needs to easily accommodate such expansion.
- Transferrable – DRCOG did not want to build special-purpose travel model software, but rather a system that could be used by many groups in the organization to run a variety of applications.
- Distributable – the system should be flexible, taking easy advantage of many hardware configurations to enhance the model's operability over the long term.

- Upgradable – the more modular the software, the easier it is to "plug in" upgraded versions of components.
- Integratable – the more standard are the software tools used to develop the system, the easier it is to integrate it with the organization's other software/business tools (for example, building web-pages that display model inputs and outputs, reading from standard relational database environments.)

At times these objectives appeared to be in tension with the need to build a system that present and future modelers at DRCOG could operate, maintain, and upgrade. Ultimately, this software approach led the project team to a design which involved development, maintenance and upgrade not *just* by travel modelers, but by IT specialists as well, with IT specialists contributing more to the "higher" level software, and travel modelers contributing more to the "lower" (more travel model-specific) code. For travel modelers to develop and work with a model built in this manner, strong programming skills were required in addition to theoretical knowledge of models. The success DRCOG modeling staff had in working with IT specialists suggests that these design decisions will prove to be practical ones over time.

Software Architecture

In some cases, prior implementations of ABMs have involved tightly integrated programs consisting of a set of nested loops, with the main procedural loop operating in a depth-first fashion over households or persons, and the individual model choice components called within the overall procedural structure. In contrast, DRCOG took what could be characterized as a dataflow approach. The model consists of a set of loosely coupled components that operate by transforming sets of input data into corresponding sets of output data. The process of running the model thus proceeds in a breadth first fashion, with each component completing the processing of all its input data before the next component starts, similar to the operational flow of older four-step travel models.

As part of this approach, with the exception of skim matrices produced and stored using a structure provided by standard travel demand modeling software, all input and output data are stored in a relational database (RDBMS). This allows the components to be decoupled from each other; any individual component simply reads its inputs from standard travel modeling software or the database and executes its specific function.

Figure 1 shows an overview of the software architecture. The actual running of the model is controlled by the *model engine.* The model engine is guided by *scenarios,* which specify which components are run, and in what order, and where inputs and outputs for each component should be found. Scenarios are specified by the user through a graphical user interface (GUI), and passed to the *model engine* by a *request dispatcher*. Input data to model components, and output data from the components, is passed between components and the database through a *data store manager*. This structure means that, while the components are expecting input data of

certain types and schemas, the exact sources and formats of the data are specified outside the component and thus are easily changeable.

Two key advantages of this approach are that the model components are independent, pluggable objects and that it is easy to run only portions of the full model. This has been very helpful during the development process because:

- Components could be developed relatively independently.
- It is easier to monitor individual components.
- If an error occurs in one component, it's not necessary to rerun the complete scenario (helpful during both development and calibration phases), and intermediate outputs can be examined in the database for clues.

In addition, this approach made it possible to spot bottlenecks in processing, and tune both individual components and the generic logit model classes.

Another significant design choice made by DRCOG was to store the model *structure* in the database (i.e., utility functions, and their terms, variables and coefficient values, nesting structure, etc.) in addition to storing model input/output data in the database. Thus each choice model component can be based on one of two generic logit model base classes (multinomial or nested), focused on evaluating the given logit model type as quickly and efficiently as possible. This has meant the implementation of the individual components could focus on handling setup of the data input and outputs and handling of special cases, such as branching or looping, required by some of the model components. It has also meant that improvements to base classes flow to all components. For example, while implementing one particular model component, it was noticed that the way distribution to multiple CPUs was being handled resulted in some inefficiency. A change was made in the base class that handled "threading", improving the performance of all the components, not just the one in question.

Model Databases

Data used and generated by travel models was divided into two categories:

- The *scenario management database*. This database holds information that identifies and differentiates one scenario from another, and all information describing the choice models' structures.
- The *data database* (figures 2 and 3.) This database holds scenario inputs (employment and households in zones, etc.) and outputs (tour and trips, their origins, destinations, modes, etc.)

Using an RDBMS for the model data repository provides many advantages, including:

- Concurrent data access/manipulation in a client-server environment – during model development and calibration, multiple team members can access the same data at the

same time. During operations, multiple component threads also can read/write data simultaneously, enhancing speed of performance.

- Data integrity - Enterprise-level relational database software is specifically built to control access to and prevent loss and corruption of large, rapidly changing datasets.
- Easy to view, query, and summarize data – relational databases, and the SQL language, also are specifically built to permit (and automate) fast summary and display of data.

As with the choice component code, an object-oriented approach to design was used for the database. Tables were defined first by abstract classification, then "sub-classed" as necessary to store specialized information. Advantages of this approach include a more robust (less "buggy") data structure, enhanced data scalability (easier to add new types of components and their data), and simpler data retrieval code (less complex SQL queries.)

Hardware Platform and Current Run Time

The IRM software is currently running on a virtual server, with Microsoft Windows Server 2003, SP2, Standard x64 Edition. The hardware is an Intel Xeon X5450 3.00 GHZ with four cores, and 3.75 GB of RAM. One iteration of the model currently requires 50 hours with this hardware configuration. While full operational hardware options are still being explored, simple upgrades could significantly improve performance. For example, upgrade of the virtual server software to permit eight core operations would cut run times approximately in half.

Lessons Learned (So Far)

Due in part to the unusual approach to software development taken by DRCOG, the team learned many lessons during the project, among them:

- The communication/learning overhead associated with bringing non-modeler IT experts into the project is very large: Modeling and IT staff spent many hours teaching each other what each needed to know to work effectively on the project.
- The hoped-for payoff from this approach was largely as expected:
    - After the foundational software was finally completed , modular code did indeed result in fast production: 27 choice models were built and tested in the final five months of software development.
    - Upgrades did prove easily distributable to all components. For example: a fast pass-by reference approach to updating person data in location choice models was developed for one location choice model, and automatically implemented for all other location choice models.
    - Data summaries are easy to create, and the data structure is manageable and understandable.
    - Expectations that model operations would be easily distributable have been realized. Examples include easy separation of the code and database between

two servers, and automatic multi-core detection and code threading to use all available cores.

- Possibly the most innovative feature of the IRM is the use of relational database software as the model's data warehouse, and this choice proved to be a good one:
    - Early fears that data exchange between the software and the RBDMS would severely harm run time did not materialize (most run time is taken up simply with "number crunching.")
    - Data management and access benefits from this approach already have been numerous and significant, and are expected to be more so in the future.
- Run times are much longer than those reported for some models (e.g., Sacramento's SACSIM model.) However, run times are affected by a variety of factors, both software and model design-related:
    - DRCOG currently uses all TAZs in the choices sets of all location choice models. Experience in other cities suggests that random selection of smaller choice sets produces faster performance.
    - DRCOG has ten times of day for highway path-building and assignment, which adds processing time in many stages.
    - The use of disaggregate logsums in some components is very time-consuming.
    - The C# language comes with a large body of managed code, and may be inherently slower than some other languages.
    - It is possible that the use of RDBMS is costing time, though input/output in other cities' ABM models also is reported to be very time consuming.

It therefore is too early to draw strong conclusions about the run time effects of the software approach taken in this model. However, it was specifically designed to take advantage of a variety of advances in computing technology (such as cloud computing, etc.).

Conclusions and Future Improvements

Travel modeling is in a period of rapid advance, and numerous upgrades to the IRM are expected to be necessary in the near future, such as:

- Development of a software tool to assist in developing point-based future year land use scenarios.
- Upgrade of the scenario management system (a surprisingly complex effort)
- Possible implementation of random choice set generation and shadow pricing for location choice models.
- Integration with mesoscopic disaggregate traffic assignment software
- Enhanced toll modeling.
- Upgrades following our current regional survey project.
- Various activities to support roll-out of the model to other users (such as consultants, the Denver area transit agency, etc.)

The project team's experience with software development for the IRM suggests that, while development using IT industry tools and methods appears to have come at some cost, it also provides an environment in which these and many other improvements can easily be made, and one which will allow the modeling system to easily take advantage of advances in hardware and software tools as they become available in the future.

Acknowledgments

The authors wish to acknowledge the following individuals who played key roles in the development of the IRM: Mark Bradley (model design and estimation of some model components), John Bowman (model design and assistance with the estimation of some model components), Arun Kuppam and Krishnan Viswanathan of Cambridge Systematics (estimation of some model components), Eric Ziering of Cambridge Systematics (consultant software task management), and Thomas Rossi, also of Cambridge Systematics (consultant project management and technical and theoretical guidance.)

Figure 1: Software Architecture

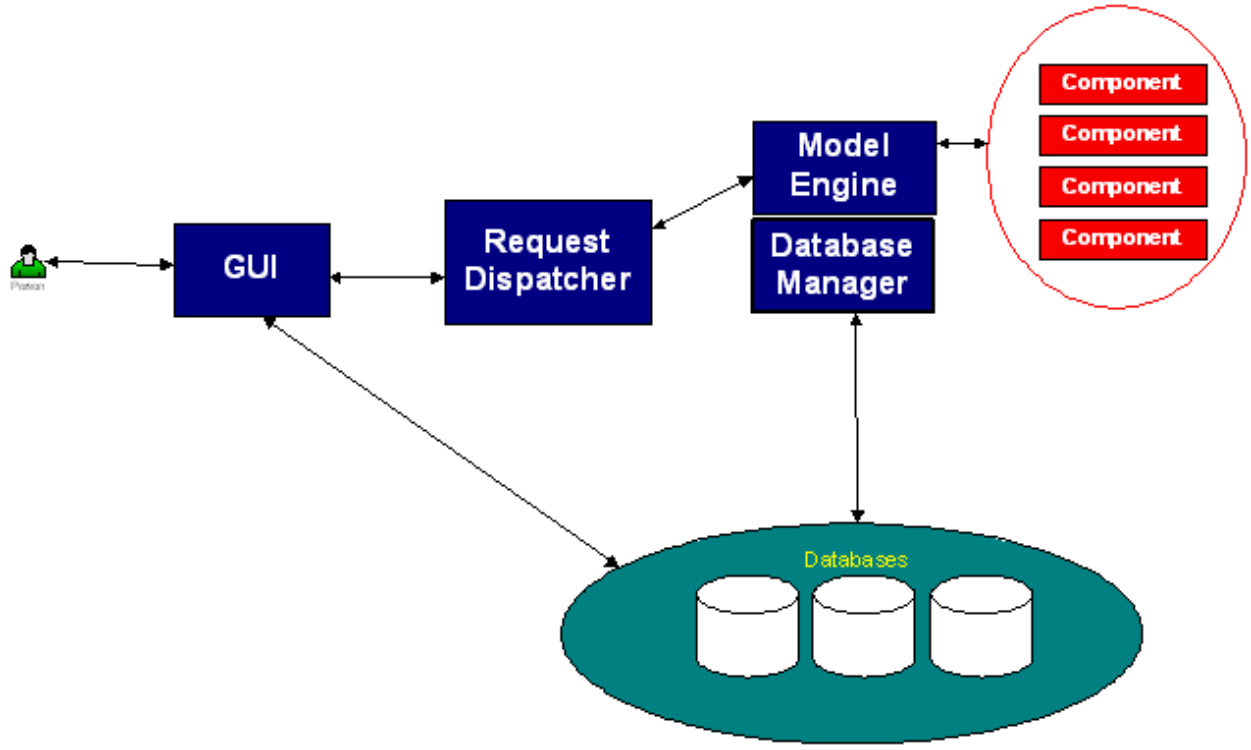

Figure 2: Zone and Point Database Diagram

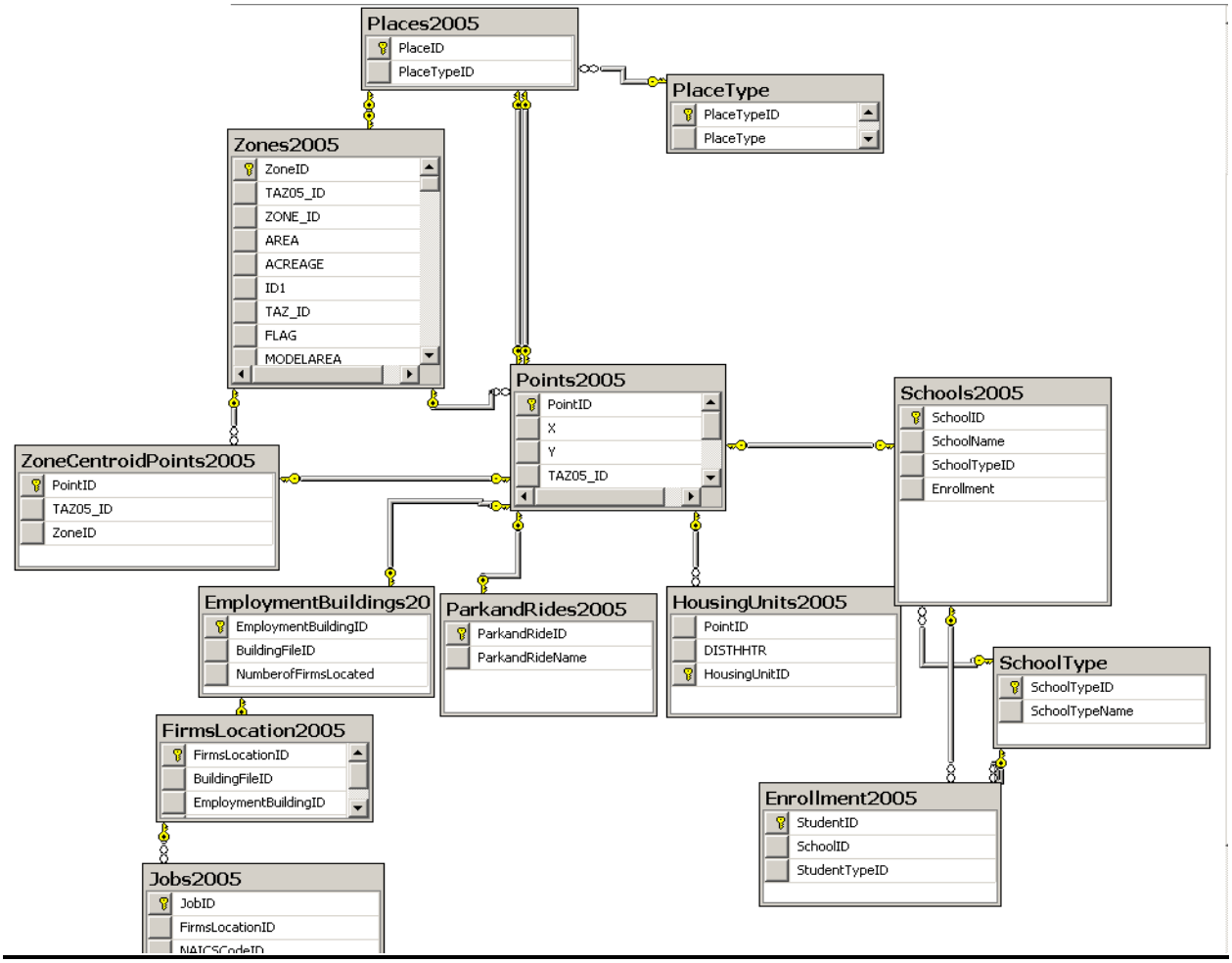

Figure 3: Tour and Trip Database Diagram