Chapter 6. Prototyping the Buildout Analysis

An important test of the PAMML services framework advocated by this paper is its ability to model the trial case presented in Chapter 3, the MassGIS buildout analysis. This empirical experiment is presented here. By referencing the problems identified in Chapter 3 and detailing how the PAMML framework addresses them, we are able to argue that PAMML not only is able to reproduce the types of analyses commonly performed by physical planners, but is also able to address the high costs of collaborative information management and processing. In this way we go beyond the basic argument, common in many disciplines, that says that the use of Web services has proven to reduce costs; therefore if we can rebuild our traditional planning support tools on top of a Web services architecture, we will naturally reduce costs in the planning discipline. This argument is persuasive, but one could argue that the planning discipline exhibits unique characteristics that prevent it from benefiting from the adoption of technologies from other fields. By explicitly addressing the information management problems exposed in the buildout analysis, we greatly strengthen the case for PAMML.

Zombie data

Recall the concept of zombie data developed earlier. This is data that are acquired from its maintainer, then used for months or years, and perhaps modified with local knowledge, with little consideration for the changes the maintainer may have made over that time. These data are dead in that they have been disconnected from their living, up-

to-date sources. Yet they are also alive because their owner is still finding them useful. The zombie data problem is at the core of the information management cost dilemma. People have come to expect applications that are lightweight and Internet-aware to have limited functionality, like the online EOEA buildout tool mentioned earlier, that aggregates statistics for multiple towns. They have been conditioned to believe sophisticated, feature-rich analysis tools like GeoVista or ArcView will depend mainly upon local data sources, and that the data management problem is external to the analysis software.

This is the key problem with MassGIS' buildout strategy. They provide excellent analysis tools, in the form of ArcView and Excel. They also provide a system for automating analytic processing in the form of ArcView and Excel macros, called the Buildout Analysis Toolkit. What they do not attend to is the data management question. This would be fine if information management was not central to the ongoing usefulness of the analysis. If the data rarely changed the cost of doing things differently would be out of proportion to the benefits. But this is not the case. Planners do want to continuously plan—they just have no feasible options to make this cost-effective. Therefore our task is to deliver the PAMML framework at reasonable costs.

In earlier chapters we discussed the issue of technology sizing. The costs of implementing a system should be heavily weighted towards the beginning of a project, when one-time funds are allocated and project advocates are energized. Ongoing costs must be as low as possible. Otherwise the technology infrastructure will disintegrate from lack of maintenance. To achieve these low costs, planning IT infrastructure must utilize general IT infrastructure as much as possible. Here we go into deeper detail,

showing an operational model of how the PAMML framework addresses the zombie data syndrome with close attention paid to the technology sizing issue. Table 1 lists a cost/effort matrix for three different data management strategies. The first, "Send data," is the most traditional, involving a data maintainer sending mailing or emailing a data set to each user. When the data changes, the entire process must be repeated. In the second strategy, "Publish data: Web site," which is the current state-of-the-art, the data maintainer uses the Web to avoid sending updates to each and every user. She instead updates one copy of the data on a Web site, then informs users so that they can download it. This strategy has proven to be a great time-saver in that the maintenance agency no longer has to handle requests for data—the Web is a self-service system—but the users' costs have not been addressed.

Table 1: Data publishing system designs

	Send data	**Publish data: Web site**	**Publish data: PAMML service**
(publisher) step 1	**Extract from operational system**	**Extract from operational system**	**Extract from operational system**
(publisher) step 2	**Copy to media**	**Copy to Web site**	**Copy to Web site**
(publisher) step 3	**Publicize updated data availability**	**Publicize updated data availability**	
(publisher) step 4	**Process data requests**		
(publisher) step 5	**Send media**	**Design-Publish Web page**	**Code-Publish WSDL, XML**
(user) step 6	**Copy from media**	**Download from Web site**	**Subscribe to data service in PAMML-enabled software**
(user) step 7	**Import into operational system**	**Import into operational system**	**Update local cache of the service**
Maintenance steps (bold)	**Repeat 1,2,3,4,5,6,7**	**Repeat 1,2,3,6,7**	**Repeat 1,2,7**

The PAMML strategy requires many of the same initial publishing efforts, but then the software takes care of ongoing updates between data publishers and users. As described in Chapter 4, the simplest data publishing technique PAMML offers is much like posting data files on a Web site. The main difference is that instead of designing an HTML Web page to complement the data file, the publisher designs a PAMML WSDL (Web Services Description Language) file and a PAMML data instance file. To make use of the data, a user "subscribes" to the data service, and from that point on, the user's software is able to create a local copy of the data set (to maximize performance), and periodically check back with the original data publisher for updates. This reduces the burden on users ***and*** publishers, minimizing the ongoing, operational cost of information management. The costs of keeping the data up to date are shifted to the software design and development stage, where they can be spread over thousands of users, instead of having thousands of users each develop their own individual solutions.

PAMML also addresses another type of zombie data problem. Data sharing often occurs without the knowledge of the official data maintainer. In addition to the data being more likely to be out of date, this leads to situations where data may be used in ways for which it was not originally intended. Addressing concerns like these motivate the data cataloging work of agencies such as the FGDC (`http://www.fgdc.gov`). In the PAMML framework, the data description file is shared (Code Listing 6-1), not the data. The new user takes this XML file and uses it to subscribe to the data service directly from the publisher. This serves two purposes. First, the new user is getting the latest version of the data. This is a nice feature, but the real significance of this strategy is that users are not passing data sets around. In effect they are passing along ***a contract*** to engage

with the data publisher. When the new user attempts to access the data, the publisher has the opportunity to decide whether or not to "do business" with that user. If the data are public and open, nothing important happens at this stage; it is simply sent to the user. However, if the data are sensitive in some way, the publisher would at this point check the user's credentials, and act accordingly. If, of course, users do not want to intentionally subvert the system, they will choose to use the PAMML framework over the old ways because, as just discussed, PAMML is cheaper and easier. And in doing so, we strengthen the contractual relationship—social, technical, or business—between data publishers and users.

Code Listing 6-1: XML instance document for Shapefile publishing

```
<ShapefileWriter srsName="EPSG:26986">
   <ShpFile dataFile="http://www.city.us/wetlands.shp"/>
   <DbfFile dataFile="http://www.city.us/wetlands.dbf"/>
   <ShxFile dataFile="http://www.city.us/wetlands.shx"/>
</ShapefileWriter>
```

In the case of an isolated data set, the idea of a contract between publisher and user seems trivial. It becomes much more significant when discussing a real model like the buildout analysis, where a number of contractual issues could arise. Is the client using server processing resources? If so, should we allow this? Are they a public agency, an individual, or a land developer? Should we charge for-profit enterprises for access? All these issues have technical solutions, and PAMML applications, by virtue of their adoption of Web services, are likely to be able to respond to them cheaply, because they can use generic authentication and security techniques designed for any Web service, instead of inventing new systems for government or planning.

The zombie data discussion started with the goal of reducing the costs and complexity (effort) of keeping data up to date. We have just shown how PAMML can solve that problem, but there is a more general issue to address. As stated earlier, there is no real difference between a data set and a model. A data set could be thought of as a concise summary of some analytic process. Therefore, any solution to the zombie data problem should also apply to analytic models. In fact, this is the case. Recall that in a PAMML framework, the user gains access to the data by subscribing to a PAMML service. The details of this subscription were contained in a PAMML XML data instance file. It would have been more accurate to call that a model instance file, because we know that PAMML does not distinguish between the two. So in the PAMML architecture—from the user's perspective—there is no difference between accessing a data set and accessing a complex model. However, from the publisher's perspective, the difference may be great. If the publisher's intent is to provide interactive access to the model, then the publisher probably needs more than a simple Web server to achieve this goal. They must first describe the model in PAMML. The simplest way to do this is to use the `GenericModel` object, which allows one to give the model a name, then describe its inputs (Figure 6-1) and outputs (Figure 6-2). Then they must implement some sort of PAMML-enabled data processing software so that users can change model parameters and run their own analyses. This more complex system is well within the capabilities of agencies, like MassGIS, who may desire them, so it is believed that PAMML offers a good match between the sophistication of agency needs and the required sophistication of their IT infrastructure.

Figure 6-1: Buildout model inputs (a representative sampling)

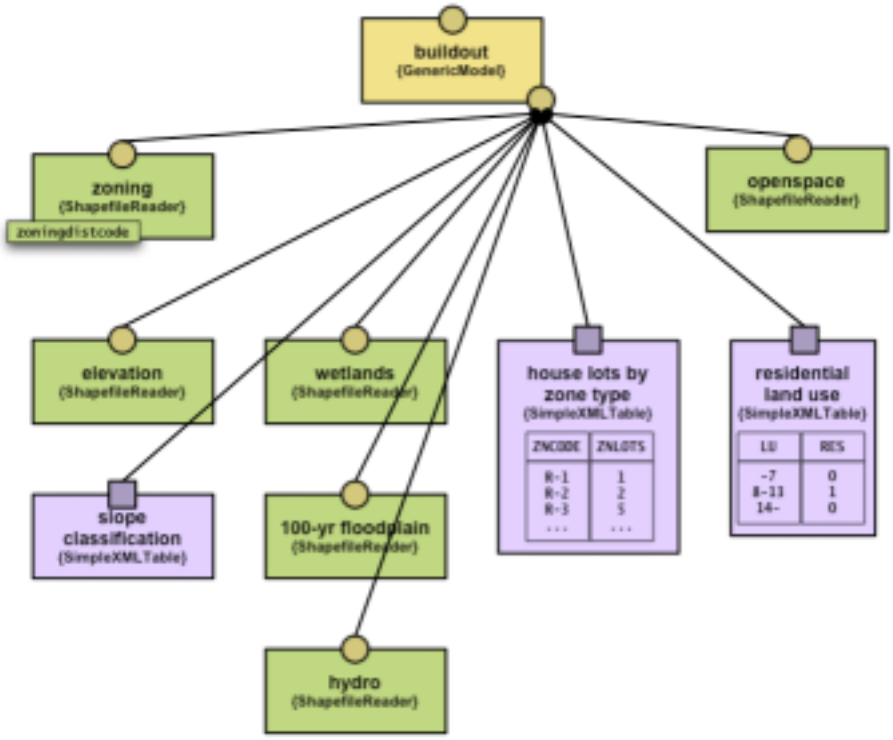
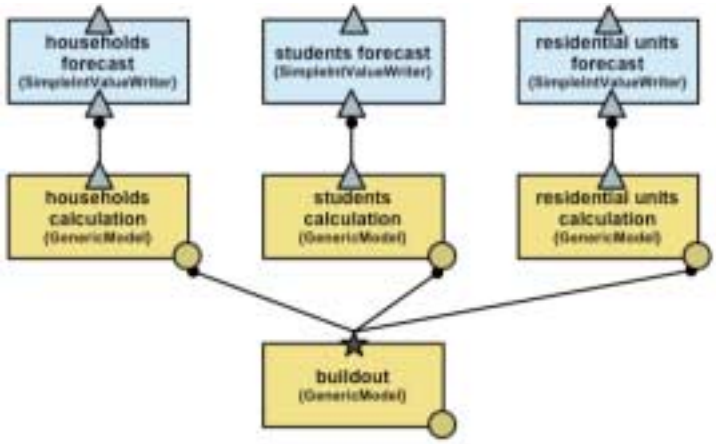


Figure 6-2: Buildout model outputs (a representative sampling)



A NOTE ON GRAPHIC CONVENTIONS: This chapter includes a number of box diagrams like those shown here. These diagrams show the process of performing analytic operations to create new data sets, which are in turn used in the next stage of operations. The diagram should be read from bottom to top, with the upper-most box being the end result of all data processing. The boxes all have a name, and an operation type—the text in curly brackets—that corresponds to an XML element in the PAMML XML schema (see Appendix A). Note the small shapes at the top and center of each box. This shape represents the type of data output by this box. A circle represents Vector data output; a square represents a table (or 2D matrix); a triangle is a single numeric value (Boolean, integer, or decimal). A star has multiple outputs. The color-coding of the boxes provides a quick visual hint relating to the type of PAMML model as well as its output data type (color differences may be difficult to distinguish in a black and white copy of this document).

Stakeholder participation

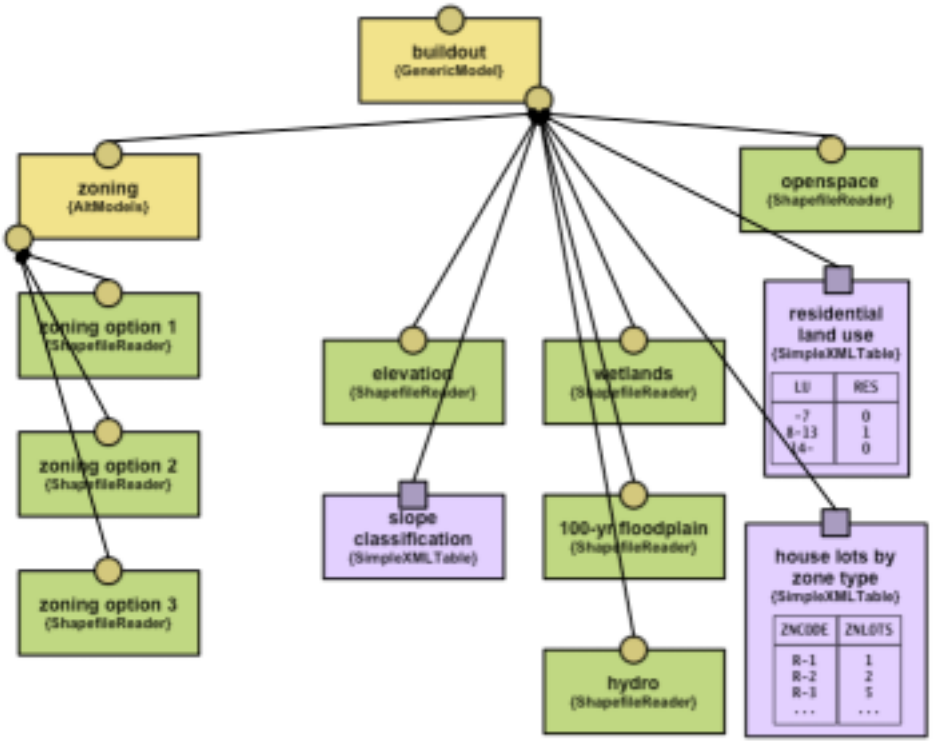
If this research did nothing but address the zombie data-model problem, it would be a success. But the PAMML framework also is able to make progress on a problem with stakeholder participation observed in the buildout analysis. There we saw a disconnect between the analysis effort and the debate, discussion, and alteration that occurs when the analysis is brought to a municipality, as discussed by Hodges (2004). While social scientists may capture this debate after the fact, this rarely happens at a time when something can be done about it. Even when there is an effort to drive new analyses, or "model runs," based on stakeholder input, this usually requires the creation of a separate system designed specifically for use in meetings, or other venues far from the analyst's workbench. Sometimes this is necessary because each model runs takes hours or days to complete, but more often it is because the modeling software is not designed to: a) be accessed outside of the office; and b) have its "data analyst" user interface be replaced by a "decision-support" user interface.

PAMML facilitates solutions to these non-performance based problems in many ways. PAMML is inherently designed to be accessed outside of one office because of its Web services roots. All operations occur via Internet protocols, whether they are limited to one computer in one office, or multiple computers scattered across the globe. The ability to apply different user interfaces to a PAMML model is even more significant. This feature is largely a result of using XML, which was designed for this purpose. The technology of how this works is discussed in more detail below.

So we see that the early design decisions to build upon standards like XML and Web services help address concerns that have often been seen as idiosyncratic of the planning

profession. But one thing we still must do ourselves is to capture public debate and discussion, and integrate it with modeling efforts. Recall that we have designed a PAMML object called `AltModel` to facilitate this. In the buildout analysis, its most obvious use would be to capture different zoning scenarios (Figure 6-3), so that zoning changes could be evaluated based on their impacts on future growth, such as changes in the number of schools required, or the increased stress on water and sewer systems.

Figure 6-3: Buildout model showing alternative zoning options



This should by no means be seen as a complete response to the stakeholder participation issue, but rather a starting point for further research. We see below how well the PAMML framework handles rich user interfaces, but the more interesting issue here is the types of information one might want to capture. For example, planners often use voting and "weighting and rating" games in participatory settings. These techniques

require an expanded information model from that presented here, but if the PAMML framework, or at least XML, is used as a starting point, the likelihood of being able to integrate the technologies is high.

Collaborative Planning

There are, of course, many more potential points of exploration and debate other than zoning regulations. For example, environmental issues are always a concern. People might debate how far away from wetlands and environmentally sensitive habitat development must be. Or they might be interested in seeing how important are the presence of multiple environmental factors in restricting the right to build. None of these issues are illustrated in Figure 6-3, because it is based on the `GenericModel`, which provides a higher level view of the analysis. We can, however, articulate these issues by modeling them in much more detail, which in turn permits a finer level of debate. Doing so brings analysis out of the modeler's workshop and into the public forum. This has always been the focus of participatory PSS, but those systems have rarely been implemented in a way that maintains a direct connection between participatory or collaborative activities, and the original analysis.

As a very basic example, recall our experience with buffer models from the previous chapter. Say that, in the buildout model, streams are protected by a 100-foot buffer zone, and this is described by a buffer model (as appears in the bottom-left corner of Figure 6-6, which will be described shortly), which is comprised of a value model and a vector data set. As full-fledged models in their own right, the buffer, the value, or the vector data could each be observed and replaced with alternatives. The result is that people with

different expertise can focus on exploring and refining different sections of the model, and this is what expert collaboration is about.

The buildout analysis was not difficult to express in PAMML, although the XML code is not easy to follow without careful study. The model is more likely to be seen using some sort of visual analysis tool, and this is how it is presented here. Remember from the earlier discussion of the buildout analysis that the general flow of the analysis follows these steps:

1. Take already developed land as-is. No redevelopment of these areas.
2. Take other areas and remove places under permanent protection from development.
3. Identify areas with partial restrictions on development.
4. Calculate maximum residential and commercial development for areas identified in step 2, and use step 3 to apply a penalty factor, arriving at a final buildout value for the area.

Figure 6-4 shows a model of step 1. Areas already developed are specified using MassGIS *landuse* GIS data and selecting out those areas whose land use identifies them as already being developed. This is the *Reclass* model named *developed, in land use database*. Note that a *Reclass* model is comprised of a vector data set, *landuse*, and a table (in purple), *MacConnell land use*. This table is used to reclassify the land use data set, mapping values of the *LU* property to values of a new property, *DEV*. In this case, the reclassification table says that for the *LU* property of *landuse*, all values equal to or less

than 7 map to a *DEV* value of 0. All values between 8 and 13, inclusive, map to a *DEV* value of 1, and so forth.

The *landuse* data is up to ten years old in some places, so it is useful to supplement it using local surveys—the *newlu* model—as well as the latest projects from developers—the *subdivisions* model. These are combined via a *Union* operation to form a model of *newly developed subdivisions*. These are in turn *Union*-ed with the older data to create a full model of developed land, which we simply call *developed*. This is the fine-grained model of developed land that the planner creating the buildout model would use. However, someone else might have no need to know all the considerations that led up to the overall conception of developed land, only the final result. In this case they would never need to look deeper than the *developed* model. At that level of detail the model looks like a vector data set with a name, which offers human users with semantic clues regarding the data set's content.[1] They would only see how that model of *developed* land was constructed if they drilled down deeper.

Figure 6-5 creates a spatial data layer containing all the various environmental conditions that could restrict development in a particular area. The final decision on how greatly they impact development is decided by human judgment, rather than an algorithm, so the purpose of this model is to do some geographic accounting. The end result of this is that the analyst has every area of the town tagged directly with its

[1] An information community could develop a better strategy for passing along semantic information other than the name of the model by adding a custom object within the *Metadata* object, which is a component of every *Model*.

environmental issues, so that a decision regarding development potential can be more easily made.

Figure 6-6 models lands that can not be developed for environmental reasons, including steep slopes, flood plains, and wetlands. These all go into a model of `partially developable` land. The notable characteristic of this model is the great diversity of primary data sources used. In Massachusetts, environmental data is usually acquired from MassGIS, who gets it from a federal agency, but performs some further processing to make it easier to use for regional work. Wetlands data may still be acquired from other sources, and three possible choices are illustrated in this model. In fact, MassGIS specifically discusses this wealth of choice for wetlands data, so its inclusion here is a necessity, not just a nice way to use the `Alternatives` model discussed in the last chapter.

Figure 6-4: PAMML model of developed land

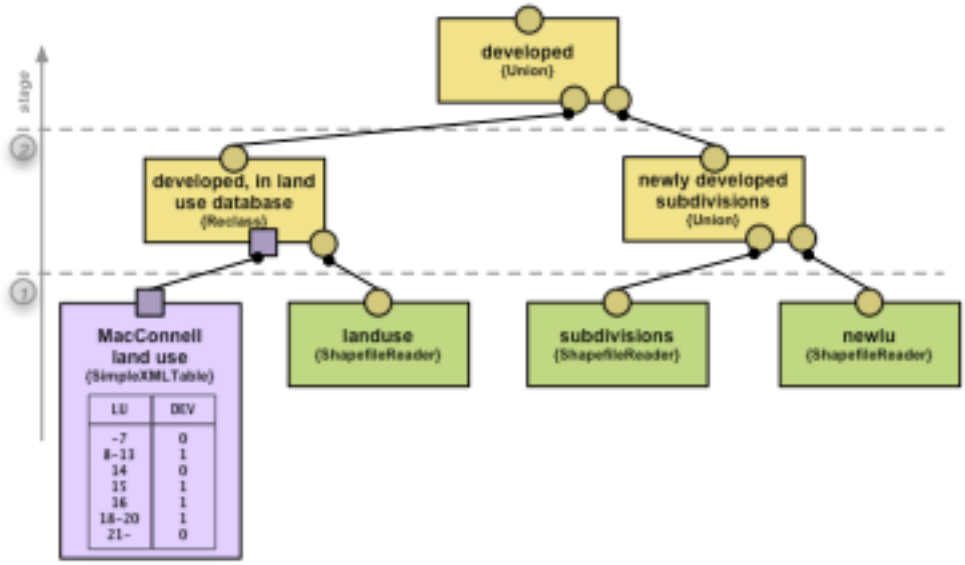


Figure 6-5: PAMML model of lands with development constraints

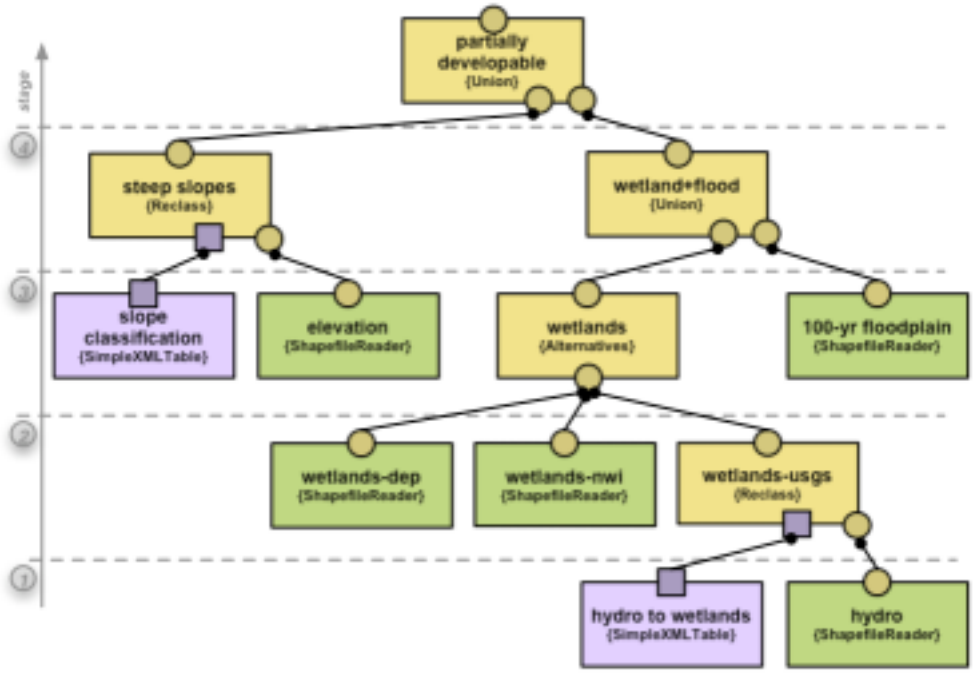


Figure 6-6: PAMML model of land that is exempt from development for environmental reasons

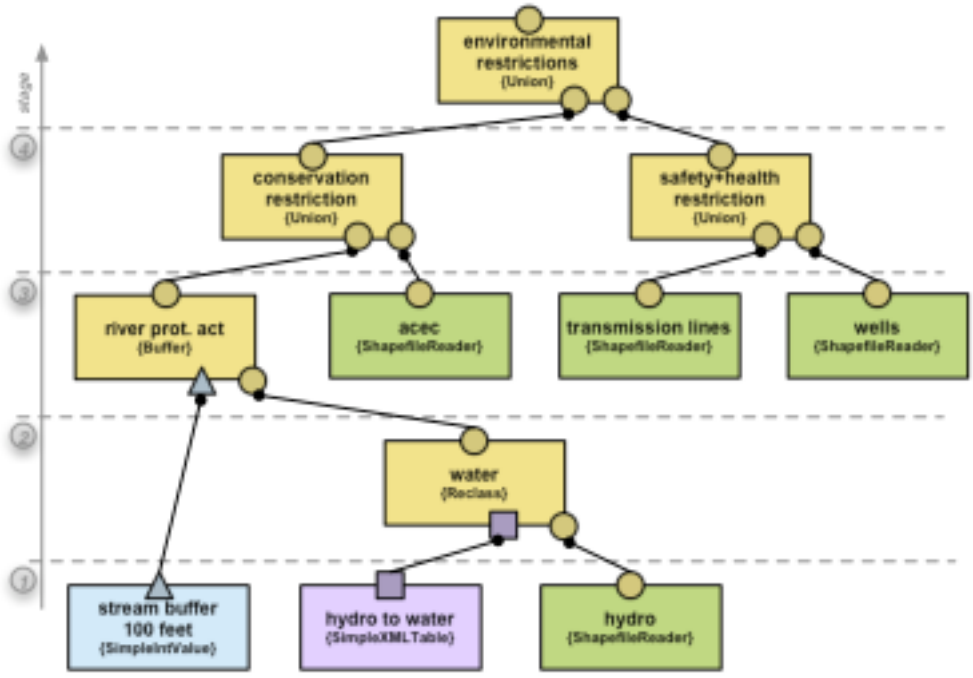


Figure 6-7: PAMML model of buildout
("partially developable", "environmental restrictions", and "developed" models are summaries of models shown above)

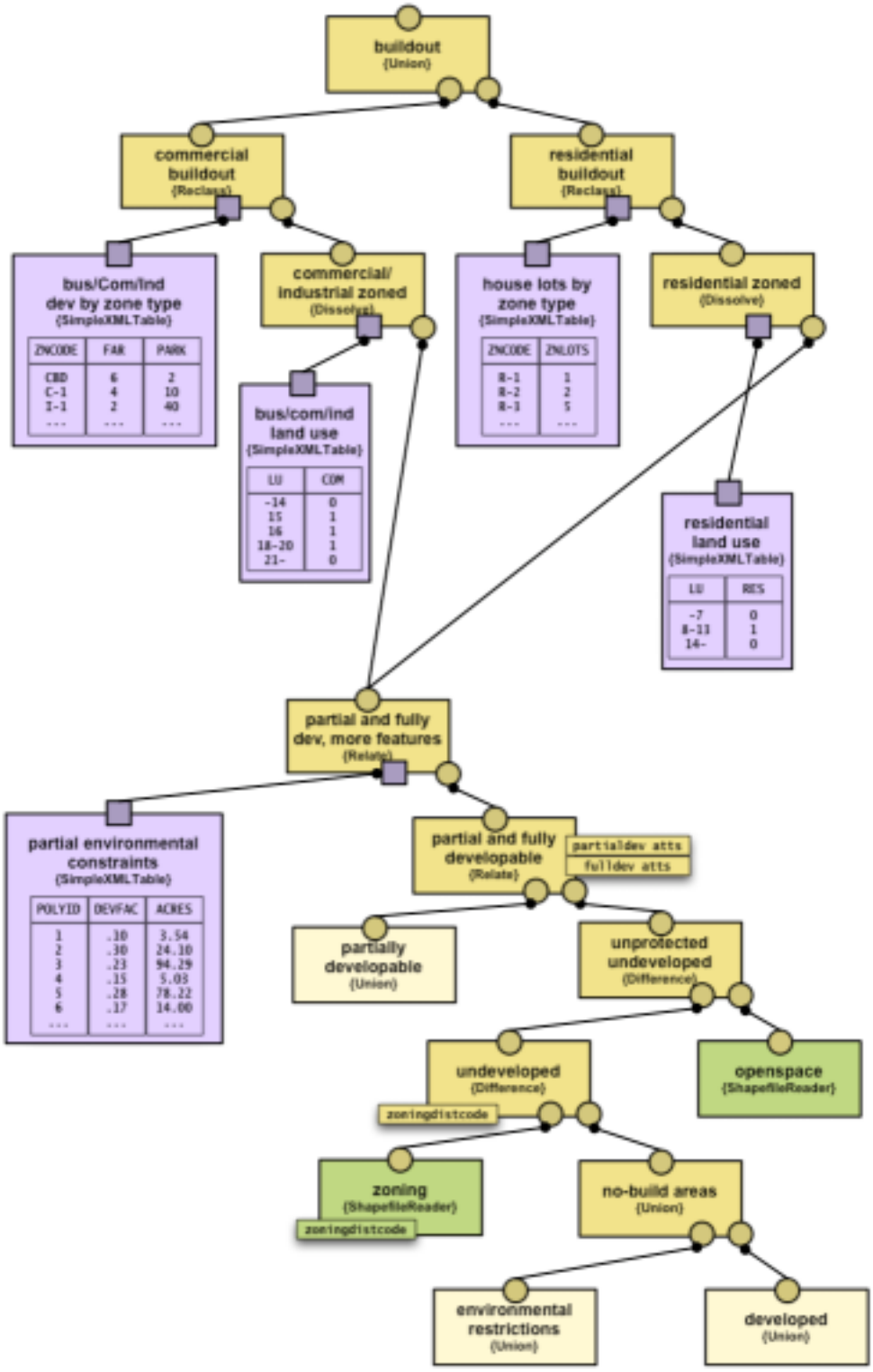


Figure 6-7 illustrates the final steps of the buildout analysis. Reading from the bottom, *developed* land and areas with full *environmental restrictions* are combined using a union model to create *no-build areas*. These areas are then subtracted from the town using the *zoning* data layer so that the zoning codes can be attached to the areas that are left in are model of *undeveloped* land. We then remove permanently protected *openspace*, to create a model of *unprotected, undeveloped* lands. Next we attach the attributes of the *partially developable* areas model to create the model, *partial and fully developable* areas.

Now we encounter the most important feature of this diagram, the reclassification, or lookup, tables. *Partial environmental constraints* is the table that an analyst would create to identify how big an impact partial constraints (from Figure 6-5) would have on development of that particular piece of land. In the MassGIS work, this step was performed in Microsoft Excel, whereas ESRI ArcView was used for the spatial operations, and this made it a bit difficult to track the analysis from start to finish. These two software programs could still be used to execute the analysis if they developed support for PAMML services, but PAMML gives us a way to formally describe the process without depending on any particular software package. This diagram also shows that this crucial stage of the analysis, being able to be represented as a simple lookup table, could easily be made available on the Web for interactive scenario generation, even if the rest of the model was more of a "black box."

Bus/com/ind land use *and residential land use* are simply used to dissolve the model of *partial and fully developable land* into residential and commercial,

because they are analyzed differently. The analyst then uses the final two lookup tables, *house lots by zone type* and *bus/com/ind dev by zone type*, to develop maximum development figures based on the density of development permitted under the zoning code. *Residential buildout* and *commercial buildout* are then merged back together to create a final, unified view of *buildout*.

Constituents want to plan continuously, not once. EOEA recognized this, and provided two avenues for further analysis. The most basic is the online application described in Chapter 3, which mainly allows a user to get aggregate statistics on multiple, neighboring communities. The more flexible option is to download all the data sets used in the analysis, and use them with one's own software (ArcView and Excel and the analysis toolkit) to create a custom buildout analysis by changing key inputs such as building setbacks, road widths, or natural resource protection buffers.

In Chapter 3 we identified the main problem with these options—they are poorly matched to their user communities. One might imagine regional planning agencies having the most need for aggregate statistics, but they are also the most likely to be planning professionals, and desire more sophisticated tools than the Web site provides. On the other hand, smaller rural and suburban towns have the most use for a system that allows them to play out scenarios based on changes to local land use regulations, yet they are unlikely to have the resources necessary to make full use of the ArcView/Excel system. And even if they did, that system is still flawed in that it exacerbates the zombie data problem. A local planner is not likely to have the time to: a) acquire and develop the skills necessary to use the ArcView/Excel system; b) work with other departments to get the latest zoning and development data; and c) find a way to share these updated data

sets with regional and state agencies. These activities must all depend upon one another if we hope to see them always performed.

The PAMML services framework does this. In order to do custom analysis, a local planner acquires the PAMML model from the state. To run the model, they either buy commercial software that understands PAMML, or they might remotely access an online suite of analysis tools that understood PAMML.[2] Either way, whether the model processed the analysis locally in commercial software, or remotely using a Web service, the core data sets would still be accessed via Web services. The local user ***could*** physically change the model to point to a local copy of the data, but ***it would be easier*** to leave the model alone, and update the original data set. This strategy updates the data for everyone (who is using the PAMML framework). Note that this framework is flexible. People can still do things the old way, but it's ***easier*** to do things right. This concept is a key design feature of PAMML in that the time and effort required to accomplish a task is aligned with the desired outcomes.

Machine-to-machine interaction

With the detailed model we have now developed, one gets a better picture of how extensive the opportunities for exploration are, but also how complex even a simple analysis like buildout can be. We have already digested the model into a diagram instead of presenting the raw PAMML XML code, and it is still complicated—not because the

[2] A state agency like MassGIS might develop and provide local planners access to such a system.

XML technology is cumbersome, but because the process of analysis is inherently quite intricate when every step is formally articulated.

We struggle to share data and collaborate on analysis in part because it is difficult to manage all these steps, and this is where the value of a Web services architecture really becomes apparent. As we observed in the buildout analysis, planning problems usually require data inputs from many sources, and the expertise of many different kinds of people. This situation implies that many different types of computing systems are involved in the solution to any problem. The Web services architecture can be thought of as a programming language for distributed, loosely coupled computers. This is in stark contrast to most programming languages, which are designed for developing software programs that will be run on a single computer on a single operating system.

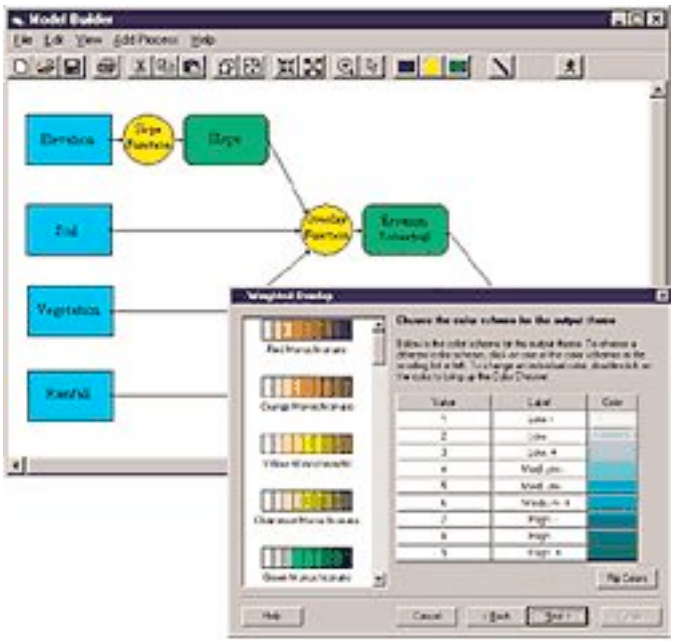
By using a technology framework designed to leverage industry-wide solutions to machine-to-machine interaction problems, we do two important things. First, we use a framework that is well-aligned with the distributed nature of organizational relationships in planning. Second, we are able to focus on planning problems instead of inventing new technology solutions from scratch. With a Web services architecture in place, we know that complex problems can be broken down into more manageable pieces, so that different people (or organizations) may develop the part of the system in which they have expertise. We have been able to do this before Web services, but it has been executed poorly, or at too high a cost. Now we have the tools needed to make machine-to-machine interaction feasible.

Interactive End Products

While the complexity of information processing may be managed through a distributed computing architecture, people must still strive to understand problems as a whole. Clearly interaction with the system must be mediated by applications and user interfaces that focus on a particular task or audience. The ability to build and interact with complex models through a visual interface is a hallmark of modern geographic information systems. Up to now, the case for PAMML has been based mainly on its importance in accurately capturing the ***process*** of information management and sharing, and through a better articulation of this process, creating the opportunity for automation and componentization. However, if we hope to ever "plan continuously," the PAMML framework must not only save time, money, and effort, but must also drive the rich visual interfaces that professional planners demand. Visual modeling and analysis interfaces are common features of commercial software. In the planning field, ESRI's ArcGIS is the most popular package. It's main interface is a map, into which data sources can be added and styled cartographically. A visual tool called ModelBuilder™ has recently been added to ArcView, allowing users to design an analysis using a wiring diagram metaphor (Figure 6-8). While ModelBuilder™ captures the modeling process in a powerful visual metaphor, it does not go beyond being a front-end to an internal scripting language. It does nothing to expand the analyst's role beyond that of the desktop, or enterprise GIS user by, for example, facilitating collaboration across users of ArcGIS, let alone other software packages. While ModelBuilder is a new product, visual model building software has been an active area of research in PSS as well as computer science in general for years. Proposals for generic enterprise modeling and analysis

toolkits can be found in abundance (Ledeczi, et. al. 1999, Delen and Benjamin 2000), but more relevant are the geospatial applications. GeoVista (Gahegan, et. al. 2002), shown in **Error! Reference source not found.**, is probably the most mature. It provides a graphic interface for spatial data analysis, exploration and visualization. A staff of researchers at the Pennsylvania State University are tasked with the software's continued development and maintenance. Visual Map Algebra (Figure 6-10) is a graphical user interface to Tomlin's ubiquitous map algebra raster analysis framework (Egenhofer 1995).

Figure 6-8: ESRI's ModelBuilder, a visual user interface to Spatial Analyst



While these systems have a multitude of useful analysis features, when viewed through the lens of this work their similarities are more striking than their differences. All of these systems have two primary characteristics, the artisan work model, and the lack of any attention to systems interoperability. The artisan model is one where tasks are accomplished by a few, skilled people in a workshop alone with their tools and materials.

The artisan strives to improve their skills and acquire new tools to produce better products. This has been translated into software as products with words in their name like “workbench” or “toolkit,” and the analysis environment and the tools are always seen to be idiosyncratic of the artisan-user. These products often have sophisticated tools allowing a user to build their own model, invent new model types, or save the description of the model for later re-use; but we do not observe an effort to share models by promulgating a standard language, or support a model structure that supports multiple users collaborating. This mindset stands in contrast to the PAMML framework, in which data are not materials, but other tools. And tools must at some level be shared, which requires systems interoperability. So a key concern of this work is to retain the useful analytic features and user interfaces of visual modeling software, while using PAMML to address those aforementioned drawbacks, that hinder progress towards reducing the costs of planning analysis.

Visual modeling

An experiment was performed to see how well PAMML would be able to integrate with the type of visual model construction environments being advocated. This is a test of the framework’s ability to appeal to traditional PSS designers and users. The task was viewed as an exercise to represent objects and their semantics in a visual environment. As a graphical user interface environment, a generic network diagramming library called JGraph (`http://www.jgraph.com`) was used. This provided a set of tools for drawing shapes, moving them around the screen, connecting them with lines, and automatically laying out network diagrams. As JGraph was developed in an object-oriented programming language, Java, it was a relatively straightforward exercise to extend

JGraph's standard graphic objects to include a fragment of PAMML XML code. It was then possible to create JGraph PAMML objects that acquired unique characteristics, such as color and shape, based on their PAMML type (Figure 6-11). Note that it was possible to fully replicate the rich graphic conventions used to illustrate the MassGIS buildout model.

A separate issue was the need to move models from their representation as XML text in a file, to visual objects on a computer screen. A number of tools were investigated that could programmatically accomplish this task. It seems that while a number of toolkits exist to automate the creation of visual interfaces to XML data, none of them were fully developed enough to use for this work. This could be because XML is not conducive to this kind of automation, but it is probably only that XML tools are still maturing; it took many years for good visual HTML editing tools to be widely available, and XML is only about five years old.

One popular mainstream application with nascent support for visual editing of XML data is Macromedia Flash MX (`http://www.macromedia.com/software/flash/`). Its roots in the Web design world (as compared to the information modeling community) are quite evident here as Flash has no way of building an interface directly from an XML Schema. It needs a concrete XML instance document for this. That technique can work for small, simple documents where every data field is always populated, such as a purchase order, but in our case, we have an extensive language in which no one model ever uses the entire vocabulary. In other words, Flash can be taught how to build a user interface for `Buffer` models, or `ShapefileWriter`s only, but it offers no tools to simplify

development of a user interface for the entire language. Hopefully the program will mature in this respect in coming years.

Figure 6-9: GeoVista Studio's Design Box, showing connected components

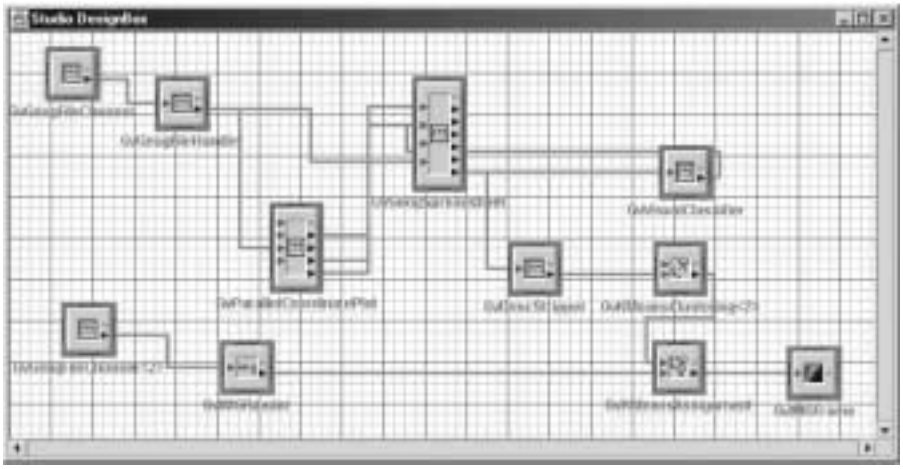
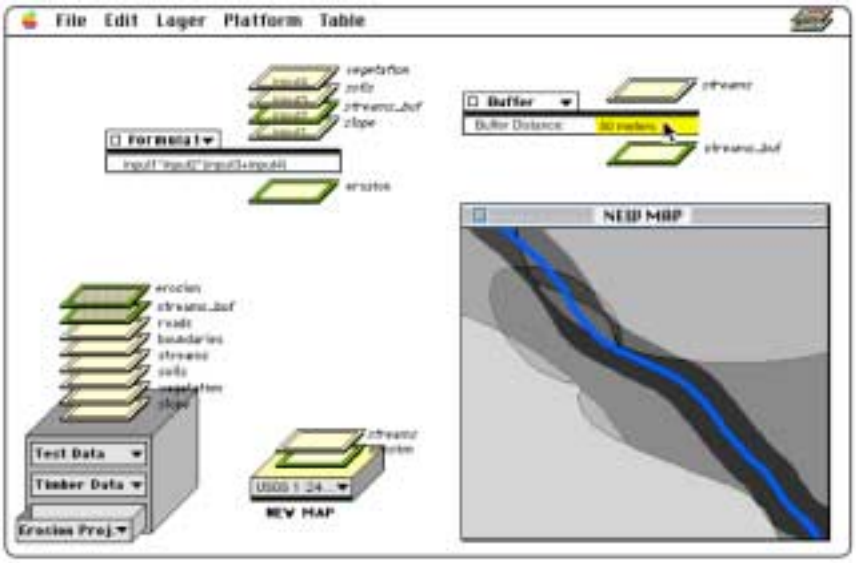


Figure 6-10: Visual Map Algebra as used in the Geographer's Toolkit



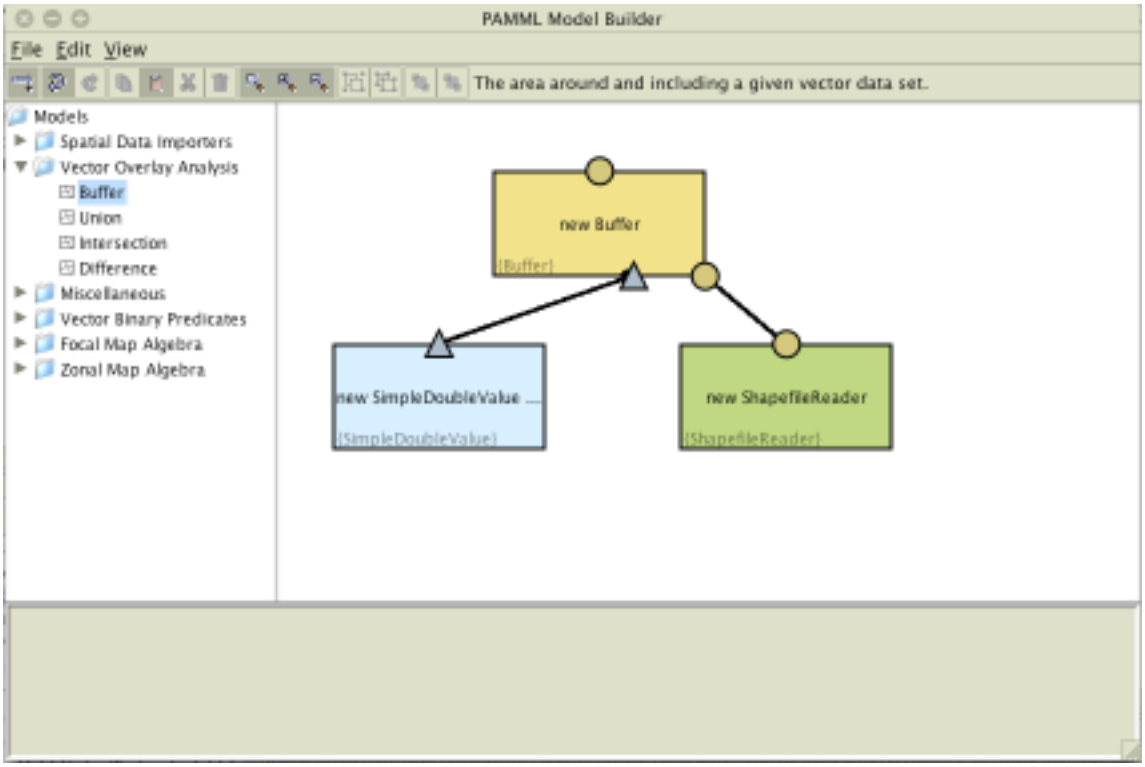
While Flash comes from the community of Web design, Sun Microsystems' Java for XML Binding (JAXB) library comes from programmers. JAXB is part of Sun's Web Services Developer Pack (`http://java.sun.com/webservices/jwsdp/index.jsp`). It is used to generate a Java version of a given XML Schema—in this case PAMML. The motivation for JAXB, and other similar applications, is that there should be a clear separation between the ***translation*** of an XML vocabulary from text to software, and the ***use*** of that vocabulary within the software program (by contrast, one could imagine a program that used the XML file as its primary data object, and any information processing that occurred in the software program would be reflected directly in the XML that it produced). JAXB was able to read PAMML XML and instantiate it as Java objects. It was then a simple matter to connect these PAMML Java objects to the JGraph objects. The library also automated the translation of PAMML back out of Java and into XML. This was an indispensable tool during the language development phase, because it allowed applications to be built while the language was still being fine-tuned. Changing the PAMML Java code was simply a matter of running a script and re-compiling the program.

While JAXB was a useful tool in the prototyping stage, it does not deal well with some of the advanced features of XML Schema.[3] Once these became important to PAMML's design, JAXB could no longer be used. While this was disappointing, the benefits of JAXB's automation features are less compelling when dealing with a stable XML Schema. It is just as easy, and more flexible, to write one's own XML processing

[3] Such as the XML Schema Instance <type> element, more commonly known as <xsi:type>, which is XML Schema's mechanism for implementing object-oriented inheritance.

and visualization routines. The failure of JAXB, and the subsequent ability to carry on without it, underlines one way in which the choice of an XML framework was a sound initial decision. Despite the relative youth of XML, no barriers were encountered that suggest that the PAMML Web services framework would impede the research and development of visual interfaces to analytic models.

Figure 6-11: PAMML modeling using JGraph and JAXB



Non-technical user interfaces

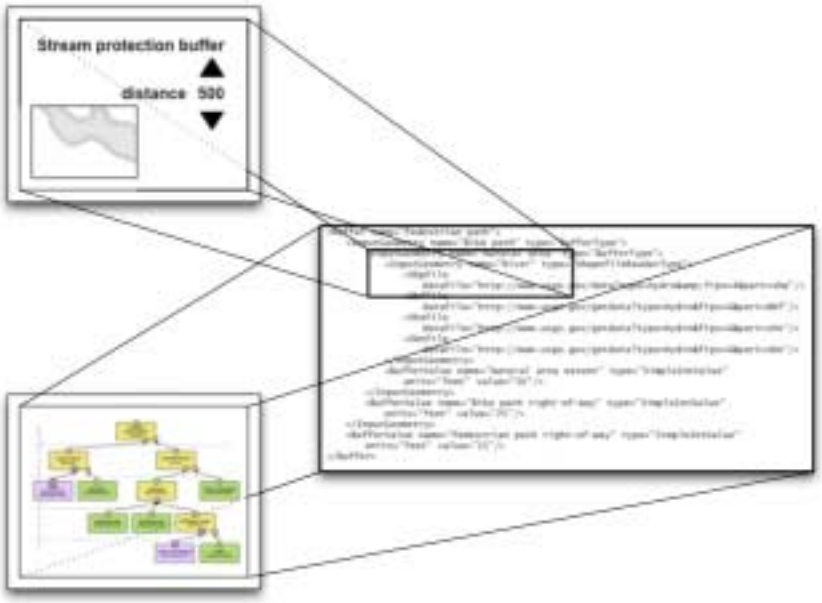
The previous discussion has covered rich, visual interface tools geared towards planners and modeling professionals. However, users with less modeling expertise must be engaged in the planning process also. This requires that our XML models take on a

much simpler interface than those just discussed. It is not surprising that PAMML can accommodate this requirement, but the way the requirement is met is extremely interesting.

The end result of the buildout analysis is the series of maps shown in chapter 3, along with a group of statistics like those listed for Sutton, MA in Figure 6-12. We know that in a PAMML framework, these statistics would be the output of models, which were described in XML. If we wanted to reproduce the Web page in Figure 6-12 from a PAMML model (which would allow the page to always display the latest projections), we could use one of a number of industry-standard XML processing tools that generate HTML code from XML. This would be cheap, not only because a host of tools already exist in commercial and open source marketplaces, but also because a host of skilled labor (Web designers) exists that can develop these applications with little additional training. Now we find our strategy of embracing mainstream technology taking us beyond direct reduction of technology costs, and into labor market efficiencies, underscoring once again the importance of integration.

Figure 6-12: Buildout analysis summary for Sutton, MA

Figure 6-13: Multiple user interfaces using the same code



The rich, visual modeling environment geared towards analysis professionals was prototyped at the front end by transforming PAMML XML into a programming language (in this case Java™), which could then be used within a traditional software development environment to develop any desired application. That user interface is still a tool whose output is a PAMML model, which describes an information processing job. This job must then be executed by an information processing engine, like a GIS system, which might reside on one computer, or be spread out among many. This strategy could be adopted to provide non-technical end users with visual interfaces as well. They key would be to present only limited sections of the model to a user, and design the interface with the user's skill level in mind. Figure 6-13 describes this scenario. An XML model is shown on the right. In the bottom left, the entire model is brought into a visual model building application like that described above. In the top left, however, we see a small part of the model (one buffer operation), presented in a much different fashion. One

variable is shown, and the value is changed by clicking on up and down arrows. In this case the user's software is smart enough to generate a preview of the buffer based on the spatial data set and the distance value chosen by the user.

We could also take an approach similar to that used above for Web page generation. There are a number of mainstream technologies available to automatically generate user interfaces from XML documents, most notably Flash and XForms (*http://www.w3.org/MarkUp/Forms/*). These are currently too simplistic or immature for developing applications for modeling professionals, but hold much promise for lighter weight, simpler applications—especially those designed for Web sites. And once again the same cost efficiencies would be realized for using mainstream technologies and a mainstream skill base. By mixing and matching the right XML-aware technologies with the right audience, we can begin to imagine how even small municipalities, with help from their regional planning agencies, might be able to provide their constituents with continuously updated, dynamic, interactive information. The completion of a housing project could trigger an update of septic loading. Or a new store opening could add congestion to a traffic model. And most importantly, any data visualization carries with it the underlying PAMML model description, so that one can imagine "copying" statistics off a Web page and pasting them into a (PAMML-aware) spreadsheet, which would not actually copy the text on the page, but the underlying XML PAMML code, so that the data does not revert to "zombie" status, and the contract between data user and provider is retained.

This chapter has presented a range of prototyping efforts, from systems design to actual software development. In each case the information management problems that are so pervasive in our profession, and are observed in the buildout analysis, were addressed. In many different ways we have seen that systems built upon the PAMML framework are likely to be cheap, scale well with organizational needs, and integrate well with mainstream technology trends. This evidence goes far towards proving that planning support systems built in this manner have a chance to avoid the systemic information management problems we observe today.