Energy sustainability through representative large-scale simulation : the logical and physical design of *xeona*

Robbie Morrison [*]

Institute for Energy Engineering, Technical University of Berlin, Germany
School of Mathematical and Computing Sciences, Victoria University, New Zealand

Tobias Wittmann and Thomas Bruckner

Institute for Energy Engineering, Technical University of Berlin, Germany

Abstract

xeona (e*x*tensible *e*ntity-oriented *o*ptimization-based *n*etwork-mediated *a*nalysis) is a new user-extensible, entity-oriented modeling environment (the software itself remains under development) whose principal role is to facilitate robust sustainability policy under conditions of uncertainty. More specifically, *xeona* is designed to study interconnected resource-transformative systems and industries, at appropriate resolution, in terms of their public interest performance. The method is particularly suited to systems which are decentralized, capacity-constrained, and volatile, and whose primary purpose is to provide physical services as opposed to tangible commodities. Energy systems typically exhibit such characteristics and the current development effort is directed toward this problem domain. The paper describes the *logical* and *physical* design of *xeona* and uses UML (unified modeling language) and C++ pseudo-code to articulate key ideas. Logical design deals with the conceptual model, whereas physical design focuses on implementation issues relative to the deployment environment. The logical design of *xeona* is fully object-oriented and combines high-resolution system optimization modeling with multi-agent simulation. *xeona* derives a number of features from the established energy system optimization modeling environment *deeco* (*d*ynamic *e*nergy, *e*missions, and *c*ost *o*ptimization). Intangible entities are central to the method and *xeona* is generally able to mimic technical control practices, unit commitment protocols, electricity price and dispatch algorithms, primary commodity and carbon markets, and simple contract negotiations. Policy-oriented investigations also require that policy interventions be explicitly captured and supported policy instruments include tax exemptions and accelerated write-downs, explicit subsidies, resource and energy taxes, permit regimes, environmental impact licensing, and incentives to promote modified technical management and/or investment. System performance under normal and extreme external events, including worst-case weather patterns, can be readily tested. Resource entities are classified using exergy analysis to ensure extensibility. Agent entities are limited to synchronous (scheduled) interactions for reasons of simplicity. The logical design enables third-party software and/or external players to be

---

[*] Author for correspondence. Email: `morrison@iet.tu-berlin.de, robbie@actrix.co.nz.`

invoked for further analytical and/or decision support. The data model is hierarchical, rather than relational, and data interchange and persistent storage utilize XML (extensible markup language). Model design and construction comprise only part of the modeling task. The way in which a particular model is applied and interpreted is of equal importance and pointers to relevant literature are given. Stakeholder acceptance is often pivotal for model-supported policy development and open source (published) software should help reduce public mistrust as well as furnish added safeguards on code and model quality.

Introduction

This paper presents a *multi-criteria decision-support modeling environment* (or model for short [1]) aimed at *resource transformative industries and activities*. The model is suitable for questions involving public policy, private investment, or combinations thereof. The emphasis in this paper is on *national public policy formation* for the purpose of eliciting *locally least-cost sustainability improvement* responses. The precise definition of sustainability is left as a modeling decision, but *biophysical metrics* are assumed. A given problem domain necessarily includes all relevant supply obligations, point-of-demand facilities, supply networks, commercial relationships, and price and non-price supply chain selection mechanisms. The problem domain of interest here is that of *energy-services provision*. The energy-services provision domain forms a relatively distinct system within the broader scheme of resource transformative industries and thereby presents natural boundaries for the purposes of modeling. The modeling environment being presented is called *xeona* : *e*xtensible *e*ntity-oriented *o*ptimization-based *n*etwork-mediated *a*nalysis.[2]

A *main objective* of the *xeona* project is to provide a problem domain simulation platform upon which public policy interventions can be tested in an *integrated manner* and against predefined *strategic outcomes* (labeled PIPC and discussed later in this section). The *xeona* project is motivated by the need to develop software tools that allow policy analysts and public interest decision-makers to confront the complexity of the systems they face. Traditional methods are failing in this regard, particularly where important aspects of the problem are not amenable to isolation, aggregation, extrapolation, and/or single criteria evaluation.

At the time of writing, *xeona* has yet to reach alpha release. All key software design decisions have been taken and much of the core functionality has been coded and reviewed. Most *entity* class hierarchies (primarily covering technologies, resources, and agents) have been finalized and trunk classes written. The process of developing specialized concrete (ordinarily useable) entity classes represents a substantial and ongoing task. These issues notwithstanding, *xeona* is presented here *as if complete* in order to simplify the writing style.

[1] Strictly speaking, a *model* is an instance of a *modeling environment*.

[2] Other roles not specifically covered in this paper but no less applicable include: (1) the evaluation of uptake prospects for energy technologies in pre-commercial form, (2) the assessment of additionalities related to Kyoto Protocol flexibility mechanisms and similar, (3) the high-level (scheme) design of energy systems and associated supervisory control protocols, and (4) the public interest analysis of commercial conduct in commodity markets, including wholesale electricity. While these roles may seem very different, the underlying simulation problem is surprisingly similar in terms of establishment.

xeona provides a numerical modeling environment upon which some selected *system of interest* can be *constructed*, *run*, and *evaluated* — and, assuming that the mode of application is *comparative analysis*, subsequently modified, re-run, and re-evaluated. The term *system of interest* denotes a selected subset of the set of resource transformative industries and activities plus their relevant contexts — such as demand obligations, weather patterns, the commercial environment, the institutional setting, and so forth. The term *constructed* alludes to the fact that an extensible entity library (XEL) within *xeona* allows the model to be rapidly populated. The term *run* indicates that the simulation, once started, is normally left to its own devices — which will comprise (1) operational dynamics *and* may include (depending on XEL support and model specification) (2) short-run commercial dynamics and (3) long-run commercial evolution through periodic commissioning and decommissioning decisions. The term *evaluated* indicates that the resulting key system performance metrics (SPM) are used by decision-makers to quantify the trade-offs and synergies that invariably accompany operational public policy change.

The comparative analysis mode of *model application and interpretation* (see above) is one of a number of model usage strategies. Model application is a relatively involved topic and not directly relevant to the aims of this paper — readers are instead directed to Morrison, Wittmann, and Bruckner (2003). Suffice to say that the system performance metrics used to capture the notion of "locally least-cost sustainability improvement" are known as *public interest performance criteria* (PIPC). The qualifier *locally* is necessary because there is no fully general least-cost sustainability problem statement and, in many cases, the application of least-cost is restricted to operational choices. Given the current state of scientific knowledge, PIPC relevant to the energy sector include: *levelized financial cost decrease* quantified using perhaps [\$] or [€], *greenhouse gas emissions reduction* in [kg CO_2-e], *depletable resource use displacement* using for instance [J exergy], and *local air pollution reduction* [various measures].

The *xeona* project forked from the *deeco* : *d*ynamic *e*nergy, *e*missions, and *c*ost *o*ptimization project[3] in early-2003, primarily to facilitate the use of agents. The main conceptual differences between *deeco* and *xeona* are the inclusion in *xeona* of processes to represent price formation and commercial relationships, both of which rely on agent entities of appropriate sophistication. Hence *xeona* combines *high-resolution optimization modeling* with *multi-agent simulation* (MAS) as indicated in Morrison *et al.* (2003).

deeco itself is well described in the literature. Technical papers with a focus on *model construction* include Groscurth, Bruckner, and Kümmel (1995) and Bruckner, Morrison, Handley, and Patterson (2003). Technical papers which also discuss *specific projects* include Bruckner, Groscurth, and Kümmel (1997), Morrison and Bruckner (2002), and Bruckner, Morrison, and Wittmann (*under review*).[4]

xeona is written in standard C++ (ISO/IEC 14882:1998 and technical corrigenda) with additional support from Boost 1.30 packages (graph containers), the Blitz++ 0.7 library (numerics), GNU GSL 1.4 package (numerics), GNU GLPK 3.0 (MILP optimization support), libxml2 (XML support), and Xerces-C++ 2.4 (XML support) as required. The

[3] The *deeco* project website URL is: `http://www.iet.tu-berlin.de/deeco`.

[4] Publications relating to *deeco* often use the term *exergy* to describe what is known in policy circles as *energy*. This paper uses the latter terminology in the interests of readability, except when discussing PIR classification.

selected compiler is GNU GCC 3.3 (g++) and the current development platform is Linux kernel 2.4. The selected *distro* is Red Hat Linux 9.0 on IA–32 (32-bit Intel-compatible hardware). C++ was chosen as the core language due to language and library features, numerical performance, support for scientific programming, perceived longevity (particularly in light of the 1998 standard), and the availability of supporting textbooks. Scripting makes use of Perl 5.7 (with the intention to migrate to Perl 6 when released). Source control relies on CVS 1.12. Code documentation uses Doxygen 1.3. And design articulation is *via* UML 1.4 using variously Microsoft Visio 10 and Umbrello 1.1.[5]

It is fair to say that *object-oriented programming* (OOP) is revolutionizing the way in which numerical models are being conceived and implemented. OOP is replacing (perhaps extending might be more tactful) procedure-oriented programming and database programming for complex applications. The impact of this change in paradigm is being felt in two related ways. First, in terms of the problem domain, the self-evident synergies between *systems modeling* and OOP are being exploited. And second, with regard to *scientific programming*, OOP (taken to include generic programming) is facilitating greater abstraction and thereby tighter and more versatile code. In both cases, the OOP paradigm requires that the dependencies between abstractions — whether related to the real-world or to mathematics — be reduced by careful design.

The *design ethos* underpinning *xeona* is to make the core model structure — the *xeona kernel* — as general as practicable and then rely on an *extensible entity library* (XEL) to effect the necessary specializations.[6] As noted earlier, the present focus of XEL development is directed toward the energy sector and includes facilities which provide energy-services through commodity use and/or passively.

The following *software design* terminology is used. *Classes* are *logical elements*, which may, in turn, be organized into *packages*. *Components* are *physical elements*, which can be *stereotyped* as source files, component libraries, and executables, as well as data files, electronic documents, and so on. Components necessarily reside on nominated hardware *nodes*. (Pilone 2003).

The craft of *software design* can be split into two related aspects (Lakos 1996). *Logical design* is largely concerned with the way in which *class hierarchies* are conceived and interact *en-masse*. *Physical design* deals with the way in which *source code* is organized through *component hierarchies*, to facilitate implementation, manage cross-file dependencies and assist deployment. Poorly-resolved cross-file dependencies can be particularly debilitating as they extend build times, impair maintenance and testing, and downgrade opportunities for code reuse. A third topic accorded specific attention is data design. *Data design* deals with cross-component data interchange and persistent storage.

In terms of higher-level *physical design*, *xeona* adopts a *script/component architecture* in which the core code is written in a fast fully-featured compiled language (namely,

[5] Most of these software components run project websites, locatable through an Internet search.

[6] The word *library* in XEL is used in its logical sense and implies no view on whether the XEL should be implemented as part of the main executable or imported from dedicated static (`.a`) or dynamic (`.so`) component libraries, or some mix thereof.

C++) and the various executable components are then "glued together" using a more agile interpreted language (in this case, Perl).[7]

In terms of *data design*, XML is used for persistent storage. Strictly speaking, XML is an enhanced technical standard for document processing founded on the XML meta-language. The meta-language is used to describe hierarchically-organized character-based data and meta-data using a series of nested tags and elements. The standard provides for, among other things, XML specializations (vocabularies) and enforcement (DTD, Schema), document processing models (DOM, SAX), and a dedicated programming language (XSL). XML differs markedly from the relational database paradigm, in which, for the latter, two-dimensional tables, and table relationships are central.

The use of a *visual language* to articulate aspects of logical and physical design can be extremely valuable. UML has become the *de facto* technical standard in this regard. For those unfamiliar with UML, Fowler and Scott (1999) provide an introduction and Pilone (2003) summarizes the syntax. UML supports a number of diagram types covering either structure or behavior. In this paper, logical design is communicated using UML class diagrams and C++ pseudo-code. Physical design is communicated *via* UML component diagrams. And data design is communicated *via* XML pseudo-code.

Much of the *terminology* of computer programming is specific to a particular language — which can give rise to confusion. This paper adopts the following convention. UML definitions form the benchmark unless C++ definitions would be more appropriate — for instance, a discussion on C++ objects might talk about *member functions* rather than *operations*. Pilone (2003) and Loudon (2003) were consulted, in the first instance, on specific usage.

Program analysis, design, and coding require good third-party references and the following C++ titles were found helpful: Lee and Tepfenhart (2000) on logical design (templates excluded), Gamma, Helm, Johnson, and Vlissides (1995) on design patterns, Dattatri (2002) on programming practice, Lischner (2003) as a language/library reference, Loudon (2003) for syntax, Robbins and Robbins (2003) on systems issues, Berryhill (2001), Breymann (2000), and C/C++ Users Journal (November 2003) on scientific programming, Sedgewick (2002) and Siek, Lee, and Lumsdiane (2002) on graph containers, and Lakos (1996) on physical design.

The *software license* for *xeona* is based on the GNU general public license (GPL) version 2, which provides for in-house use and development but requires that binaries (executables and libraries) be distributed with unencumbered source code which also maintains the GPL. The use of an *open source software* (OSS) license is particularly indicated for public policy modeling, if only because the code and documentation can be potentially checked and run by citizens, NGOs, and business stakeholders. More specifically, the act of publishing code and data can help temper claims regarding model performance and applicability. OSS also allows interested programmers to engage and contribute, perhaps to the point where a development community emerges.

[7] A *compiled language* undergoes compilation and linking prior to execution. This allows the compiler to "optimize" the resultant machine code by identifying and discarding unnecessary CPU instructions. In addition, there is no run-time overhead associated with translating the source code. An *interpreted language* undergoes line-by-line interpretation at run-time. The interpreter has limited opportunity for optimization and the processing of source code takes extra time. In passing, Perl uses on-the-fly compilation rather than strict interpretation.

There are relatively few software projects related to *xeona* and reported in the literature. MIND (was MODEST) from Linköping Institute of Technology, Sweden (Gong, Karlsson, and Söderström 2002), TOP-Energy (was EUSEBIA) from RWTH Aachen, Germany, and work at Surrey University, United Kingdom (Shang and Kokossis 2002) have similarities in terms of conception and resolution, but are aimed at private (single firm) decision problems. The Asian Pacific Integrated Model bottom-up energy module (AIM/Enduse) from the National Institute for Environmental Studies, Japan is somewhat akin and is specifically targeted at national policy support (Kainuma, Matsuoka, and Morita 2003). Agent-based models are under development for public and private policy analysis within the electricity sector, for instance, Bower, Bunn, and Wattendrup (2001). *xeona* also shares some commonality with large-scale urban planning simulations, with one such example being UrbanSim (Noth, Borning, and Waddell 2003).

The direction taken by *xeona* (and also MIND and TOP-Energy) is in marked contrast with the operations research (OR) approach which has long characterized public policy energy systems modeling (Weinhardt *et al.* 2000). The classic OR approach adopts a database plus solver architecture, as typified by MESAP from IER, Stuttgart, Germany (Remme, Goldstein, Schellmann, and Schlenzig 2001).

The remainder of this paper is structured by section as follows: §1 discusses overarching programming issues, §2 provides information on the underlying model, §3–5 examine aspects of the design of *xeona*, §6 looks at testing, §7 provides an outlook, and §8 furnishes conclusions. An academic paper cannot replicate the information contained in hundreds of pages of design notes, diagrams, pseudo-code, source code, program comments, and formal documentation, and, as a consequence, §3–5 provide only an indication of the approach taken.

1 Programming paradigms

A numerical model relies heavily on the programming paradigms and techniques which inform its central design. This section provides some background on the paradigms and techniques on which *xeona* is founded.

1.1 Object-oriented programming

The term *object-oriented programming* (OOP) can cover a range of language and library facilities. This paper restricts usage of the term to object-based languages which support: (1) classes, inheritance, and late-binding of operations (type polymorphism), (2) inter-object relationship stronger than simple communication, namely aggregation, composition, and friendship (privileged access), and (3) parameterized functions and classes (parametric polymorphism).[8] These requirements encompass C++, but Java currently lacks feature three and Smalltalk lacks feature two. Visual Basic and procedural languages such as C and Fortran fall short on all grounds.

The concept of *generic programming* (GP) is broached by point three (above). GP allows foundation *data structures* (typically container classes) and *algorithms* (such as searching, sorting, and combinatorics) to be implemented separately and then combined as required in client code. This separation is achieved through careful abstraction of the

[8] A fourth category of declarative semantics could have been added. However, no existing language would comply, but R++, a rule-based extension to C++, is under development.

data structure/algorithm interface and is facilitated in C++ by templates and operator overloading (Siek *et al.* 2002). More prosaically, templates enable type-independent functions and classes to be written and can also be used to confirm code integrity at compile-time (Lischner 2003). Strictly speaking, object-orientation and genericity are separate issues, but they complement well and support for templates is sometimes considered a necessary ingredient of OOP.

1.2 Scientific programming

Some observations on *scientific programming* are in order because *xeona* relies on developments in this area. The established *procedure-oriented programming* paradigm encourages numerical models to be defined as sets of equations which are submitted for solution through function call-and-return semantics. Fortran, as one example, supports matrix algebra with limited finesse, but otherwise data abstraction remains restricted and individual solvers are required for each data type (Berryhill 2001). The OOP paradigm allows for far greater abstraction within the realm of scientific programming. Advanced data types organize and protect data and provide clean interfaces. Operator overloading enables a more natural syntax. Inheritance and late-binding leverage commonality amongst data types whilst retaining differentiated behaviors. Templates support type-independent algorithms and help reduce coding effort. And, GP techniques are being used to implement powerful high-level numerics libraries (for instance, the C++ standard template library).[9] As Roussel and Roussel (2003, p18) observe "the result is tight readable code that is easy to maintain and extend".

1.3 Large-scale simulation

The OOP paradigm also facilitates another approach to numerical model construction — one qualitatively different from the abstract mathematical expressions and concise scientific programming just reviewed. OOP is the programming style of choice for building *large-scale simulations*. Large-scale simulations are formulated in terms of complex entities interacting over time. These entities are mostly represented in the program domain as objects and their interactions are mediated through their public interfaces. Class hierarchies can leverage commonalities within sets of entities. Such models are typically triggered and left to evolve of their own accord. If the entities possess some level of autonomy and intelligence, they are termed *agents*. Although within *xeona*, less sentient entities are also present (for instance, collections of engineering plant together with their supervisory control practices). The inclusion of sophisticated agents gives rise to the next topic.

1.4 Multi-agent systems

The field of *multi-agent simulation* (MAS) is relatively new in the context of mainstream programming. Problem domains now include areas as diverse as theoretical economics, urban space and transport planning, distributed computing, and international negotiations.

Sycara (2003) lists the characteristics of multi-agent systems thus: "(1) each agent has incomplete information or capabilities for solving the problem and, thus, has a limited

[9] There does not appear to be as quick a response in the specialist area of mathematical programming, but it should be recalled that GP is a relatively new endeavor.

viewpoint, (2) there is no system global control, (3) data [is] decentralized, and (4) computation is asynchronous."

Bower *et al.* (2001, p992) describe the use of agents to model strategic behavior in liberalized electricity markets in Germany. More specifically, Bower *et al.* note that, in this context, MAS: (1) supports bottom-up commercial behavior over time and thus avoids the need to posit a static top-down (and thereby allocatively efficient) price equilibrium, (2) allows agents to adapt, (3) accepts that agents have limited processing capacity, and (4) allows macro complexity to emerge from bottom-up decision-making. Point one means that electricity price formation is not predicated on generation offers being necessarily marginal cost — in other words, competition deficiency in the right circumstances is supported.

The agents within *xeona* are not as autonomous as those described by Sycara (2003), but nor do they need to be. These agents often submit to some form of *club control* (in the economics sense) — as they would do in reality. In this respect, the agents are more closely linked to the agents described by Bower *et al.* (2001). For instance, wholesale electricity market participants yield to a *virtual auctioneer* — duly encapsulated as a pricing and dispatch algorithm — and then bid at mandated intervals. This then allows the use of simpler synchronous computation.[10] And unlike pure MAS, *xeona* also makes significant use of scientific programming constructs. For instance, MILP (mixed integer linear program) optimization solvers are used to drive price discovery and/or single-operator unit commitment.

1.5 Script/component architecture

In this paper, the term *script/component architecture* refers to software construction in which an interpreted script is used to invoke and manage the required executable components (Beazley and Lomdahl 1997). *xeona* adopts this approach and, in the first instance, makes use of the filesystem for XML data exchange. The strategy provides developers and motivated users with increased flexibility — for instance, model run loops can be set up, whereby new models are generated and run in light of prior results (see also, Gong 2003). On the down side, XML parsing is required in the scripts as well as in the executable components. The approach also makes it easier to write graphical user interfaces without the overhead of systems programming (Ousterhout 1998). Script/component architecture falls within the larger field of *mixed language programming*, whereby classes and functions placed in compiled dynamic (`.so`) libraries are accessed directly from scripts (Tomson 2004).

1.6 Heterogeneous model construction

A script/component architecture can also facilitate *heterogeneous model construction*. Under this development strategy, so-called *multi-models* are built from smaller discrete executable models (Jaeger *et al.* 1999). A climate policy multi-model might, for instance, be built using individual reduced-form economic and climate system models and then solved progressively. Riahi and Roehrl (2000) describe such a strategy centered on the energy model MESSAGE. Notwithstanding, Barker (1998) is critical of some economic multi-models on the grounds that the coupling between components is not sufficiently deep to capture the underlying dynamics. Multi-models require that a

[10] Asynchronous interaction is often implemented using *state machines*, which then adds a further level of design complexity (Lee and Tepfenhart 2000).

number of conceptual, concurrency, communication latency, stability and related issues be resolved. Data is normally streamed between components and the model architecture may rely variously on operating system constructs, distributed computing protocols, or even a parallel virtual machine abstraction (Robbins and Robbins 2003). Multi-modeling is attracting interest because it facilitates component model reuse, has the potential to reduce development times, and is suited to multi-node computing systems. Multi-models normally rely on two related areas of computer science, namely concurrent programming (for the purpose of exploiting a multi-processor system rather than servicing an interactive user) and distributed computing.

1.7 Concurrent programming

Concurrent programming is primarily a logical construct involving the management of program control flow and system resourcing. In the context of numerics, concurrent programming requires that computationally intensive algorithms be *parallelizable*. C++ itself does not provide native support for concurrent programming, but class designs can incorporate thread-safety and third-party kernel-level multi-threading libraries are available (Lischner 2003, Robbins and Robbins 2003). For non-interactive models like *xeona* (calls to human decision-making aside), concurrent programming is only useful when considering multi-processor or multi-node environments. *xeona* is written in unthreaded form for reasons of algorithm design as well as coding simplicity. Groscurth and Kress (1998) earlier examined the feasibility of decomposing the computationally-expensive optimization task within *deeco* for parallel solution, but the proposed approach might also have masked important network effects and was not pursued. *Distributed computing* is the physical construct associated with concurrent programming over multi-node systems. Casting ahead, however, it is more likely that software like *xeona* will be run on multi-processor workstations (with perhaps 8 CPUs) than across complex distributed computing architectures. Moreover, parallelization might remain confined to third-party numerics libraries with little or no impact on the design of client code.

1.8 Closure

The various themes in this section, taken collectively, are yielding a new generation of systems-based policy models. This emerging category of model was described in the title of this paper as *representative large-scale simulation*, although we have used the term *entity-oriented modeling* in earlier publications. The central defining feature of large-scale simulation is that the treatment of system behavior is restricted to the kinds of dispersed real-time decision processes seen in practice. In contrast, *system dynamics* models are built up using general expressions of behavior, established either analytically, inferred through observation, or guesstimated and then checked for sensitivity.[11] The second defining feature of advanced large-scale simulation is the use of embedded optimization (and/or related heuristics) to drive aspects of system progress. Again this practice is ideally limited to situations where such techniques either mimic reality or could do so at some future point. And again in contrast, system dynamics does not support endogenous optimization, although individual scenarios can be prescriptively tuned using parameter optimization and pseudo-optimization

[11] Similar issues occur within engineering simulation whereby *component-oriented model* generators (for instance, modelica) and *block model* generators (for instance, Simulink) differ in strategy. The problem domains often overlap, in which case the choice of approach may be guided by issues like modeling convenience.

procedures.[12] In passing, the established *single-actor mathematical programming* approach, widely applied to national energy capacity planning in the past, is becoming less applicable as market liberalization proceeds, technical diversity increases, sustainable technologies and system outcomes are sought, and the system itself becomes necessarily more volatile.

This new category of model has the potential to be substantially more representative than previous methods. But the resultant models are also "data hungry" and require a commensurate increase in data collection effort. Even so, systems-based model types are likely to vary in their intrinsic sensitivity to data quality. Large-scale simulation models with their focus on structure may, with skilled use, be better placed to accommodate missing and inconsistent data than other methods. This view is posited on the less abstract nature of the causal relationships which underpin simulation modeling.

2 Description of *xeona*

Matters of *scope* and *resolution* are central to systems-based modeling. *xeona* is suited to problems ranging in *spatial scope* from stand-alone facilities to a meta-national systems (for instance, the European Union). This scale range may come as some surprise. But the underlying numerics are fundamentally scale-independent — even though the entity characterizations are not likely to be so. In terms of *temporal resolution*, hourly time steps are often appropriate. And in terms of *temporal scope* (or modeling horizon), simulations may span a representative year or a full decade, depending on the mode of application. The minimum *topological resolution* (concerning technical connectivity and commercial association, as discussed shortly) is best discerned from knowledge of the *network dynamics* likely to arise in practice (Morrison and Bruckner 2002). The wholesale electricity price spikes seen in New Zealand, California, and elsewhere during times of electricity system stress provide a vivid example of one form of network effect in action. In most cases, it is neither necessary nor desirable to model every element within a real system.

Fundamentally, the model concerns the *allocation of resource flows*, such that supply obligations are met or, as appropriate, purposeful rationing is enacted. Resources are duly divided into physical and instrumental. *Physical resources* derive from physical constructs (examples include natural gas and precipitation). *Instrumental resources* derive from institutional constructs (examples include carbon permits and funds). Together these are known as PIR (physical and instrumental resources). Hence, PIR flow-rates form the primary control variables within *xeona*. For instance, capital investment can be treated as a decision to commit funds.

The design of *xeona* relies heavily on the concept of *entities*. The idea is an extension of that found in entity-relationship-attribute (ERA) modeling (Illingworth 1996). By definition, *problem domain entities* and *program domain objects* are the real-world and run-time interpretations of each other. In practice, not all identified real-world entities map to run-time objects and *visa versa*. Similarities among entities are used to construct *class hierarchies* (implemented using inheritance and late-binding operations). Structural hierarchy is a powerful organizing principle. Entities may be either *tangible*

[12] The term *pseudo-optimization* refers to techniques which approximate optimization. Sophistication ranges from simple parameter scans to genetic algorithms. It should be noted that large-scale simulation is also amenable to such techniques.

(examples include engineering plant and market participants) or *intangible* (examples include unit commitment procedures and price discovery mechanisms).

Real-world *relationships* between tangible entities give rise to the notion of *networks*. Two such networks are central to *xeona* and these networks are duly mapped to dynamic graph instances (container objects) in the program domain. One covers the set of *potential commercial associations* between agents and is called the PCA network (this entity can also support non-commercial associations if need be). And the other covers the PIR connectivity between plant and other PIR entities and is called the PIR network. The PIR network is *directed* in the sense that flow orientation must be specified and observed.

The real-world also contains *protocols* to select the "best" combination of locally available plant and other PIR processing entities to meet protocol-external supply obligations at any given point.[13] Protocols too need to be represented in the program domain in suitable form, possibly as dedicated objects offering services where appropriate.[14] For a facilities operator, the protocol might yield an *optimal unit commitment*. And for a grid-mediated market covering wholesale electricity trading, the protocol might be a *nodal auction* (Outhred and Kaye 1996, Philpott 1999). Both the examples given require numerical optimization and *xeona* duly bundles an MILP solver. But protocol entities are not forced to use this solver and developers can elect to provide solver functionality within their own XEL packages. Moreover, XEL packages can invoke external applications directly or recursively call *xeona* itself, to support agent-specific analysis.

XEL packages can also request input, usually concerning investment and decommissioning decisions, from human subjects. The presence of human cognition in the modeling loop necessarily introduces concepts from *experimental economics*.

A particular challenge regarding the design of *xeona* is the representation of feasible *public policy measures* within the model, preferably as entities. Such measures might include: carbon taxation, cap-and-trade carbon permit schemes, technology or fuel-specific tax relief, and air pollution standards. Space considerations preclude further treatment here, instead refer to Morrison *et al.* (2003) (fig. 3 in particular) which provides a comprehensive breakdown with examples.

The preceding discussion highlights two of the central *design tenets* of *xeona*. First, that real-world *items*, including plant, resources, and agents — defined as *tangible entities* — should map directly to program domain objects where appropriate (whilst noting that compromises in the interests of model rationalization will be required). And second, that real-world *procedures*, including unit commitment and price discovery protocols — defined as *intangible entities* — should similarly be represented within the model (again, simplifications in the interests of numerical performance may be needed). These design principles gave rise to the term *entity-oriented modeling*.

The identification of cogent class hierarchies and the idea of class extensibility are central to OOP. One advantage of using OOP for *xeona* is that the agent characterizations can start life in relatively simple form and can then be improved and

[13] These are described as *subsystem coordination protocols* (SCP) in Morrison *et al.* (2003).

[14] Entities which primarily provide services can be implemented as function objects or *functors*, which instantiate from classes that overload the function call operator `operator()`.

extended as experience builds, resources allow, and/or research questions dictate, without disrupting the overall structure of the codebase.

A *xeona* model instance usually constitutes a *scenario* and is intended to examine previous, existing, or future system configurations in light of normal and extreme external events (Morrison *et al.* 2003). Hence quite a range of actual and speculative circumstances can be investigated, including "what if" style propositions.

The rest of this section looks at the specific interplay between problem-domain issues, numerical tractability, and software design.

The classification of physical energy resources (examples of which include natural gas, electricity, and steam) is undertaken using insights from *exergy analysis* (Bejan, Tsatsaronis, and Moran 1996, Bejan 1997). The details of a resource assessment depend on whether the resource is in a *flow* or *stock* context.[15]

Much of the formulation of *xeona* revolves around the interplay between intensive and extensive variables and the application of certain simplifying assumptions (Bruckner *et al.* 2003). In the first instance, *extensive variables* scale directly with system size whereas *intensive variables* are scale invariant (for instance, generation capacity and reticulation voltage, respectively).[16] By default, *xeona* makes use of the approximation that key intensive variables can first be set by mimicking real-world (supervisory and direct) control practices and then held constant *while* the extensive flow variables are determined using PIR routing protocols. Intensity-setting takes place on subgraphs which cover the jurisdiction of the original real-world control system.[17] The PIR routing protocols often, but not necessarily, rely on explicit optimization. It is also possible to introduce local iterative solutions, should the two step approach just indicated be unsuitable.

Individual *plant characterizations* describe how plant act with the PIR network. These characterizations may also be subject to numerical restrictions in order to guarantee stable PIR routing results (Bruckner *et al.* 2003). Such restrictions often involve the need to maintain mathematical linearity between certain variables.[18]

Wholesale electricity is priced using *nodal pricing methods* in a number of jurisdictions around the world. Under the arrangements in New Zealand, participants (normally only generators, as demanders typically use and pay) offer capacity in 30 minute blocks. Individual generators usually prepare their bids using in-house "stack" models. The high voltage transmission operator submits grid status information. And a linear program (LP) calculates an optimal dispatch set — that is, a dispatch set which minimizes the demand weighted price of supply. The actual scheme is somewhat more involved but this description will suffice for the purposes of this paper.

---

[15] Technically, energetic and exergetic evaluation in flow or stock context, relate to extensions of enthalpy and internal energy and flow and nonflow exergy, respectively.

[16] The definitions of intensive and extensive variables derive from a *simple system* in local thermodynamic equilibrium. Normally the planes or compartments of measurement are not strictly simple systems and it is therefore necessary to apply the *bulk properties assumption* in flow and stock contexts respectively when establishing this model (Bejan 1997).

[17] The procedure is called *influential attribute setting* (IAS) in Bruckner *et al.* (2003).

[18] This and the preceding paragraph cover assumptions known more formally as the *state orthogonality assumption* and the *convexity assumption*, respectively.

In term of *flow of funds* within the model, commercial and domestic facilities attract, as the case may be, capital expenditure, duty-dependant payments and receipts, standing payments and receipts, and salvage income. Commercial agents within *xeona* use risk-adjusted *commercial discount rates* (say 25% *pa*), as opposed to social discount rates (say 5% *pa*), because this approach better captures the reality of their decision-making. Work on characterizing domestic agent decision processes will be reported in a future paper.

3 Physical design

Physical design involves the interrelationships between software components starting with individual source files (see section 6) and working up. This section examines the way in which higher level components are organized. *xeona* use a script/component architecture, based around the command-line interface (CLI) of the core executable. Data preparation and interpretation tools are not discussed directly in this paper, but are indicated in the outlook (see section 7). Figure 1 depicts the relationship between the script and the key components.

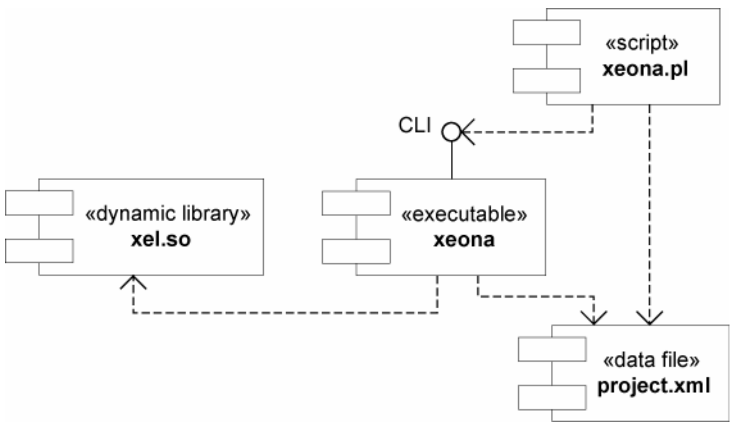

***Figure 1**: The high-level physical design of* xeona *showing the script/component architecture used. The dynamic (or run-time) library is optional and may remain in-house — however, if distributed, the shared object (`.so`) must be accompanied by source code. Deployment would normally be on a single node although calls to third-party programs on other nodes are envisaged.*

Another aspect of physical design is the way in which call-outs to third-party software are dealt with. The simplest method is to invoke `int system(char* command)` and exchange data *via* the filesystem. More complex schemes could make use of other operating system constructs such as UNIX pipes, shared memory, other forms of IPC (inter-process communication), network services, or web services, depending on the external application and its location (Robbins and Robbins 2003).

4 Logical design

Logical design can be captured from two principal vantage points: *static structure* and *run-time behavior*. These can be articulated using UML *class diagrams* and *sequence diagrams*, respectively. In passing, UML *use case diagrams* have no particular role because *xeona*, at this point, is not user-interactive.

Space considerations mean that only a brief indication of the logical design behind *xeona* can be given. The following example shows the use of inheritance hierarchies to reduced code duplication and enforce type safety. Type safety is a language feature which prevents class instances (objects) from being used in unintended contexts and is invaluable when constructing models as complex as those supported by *xeona*.

Table 1 shows the core hierarchy for PIR. This same hierarchy is also repeated in figure 2 using UML. As noted earlier, this particular hierarchy is informed by concepts from exergy analysis. Similar taxonomies have also been developed for plant, for agents, and for other families of entities. Listing 1 indicates how class declarations and inheritance might be used in practice and how objects are created.

Category	Sub-category	Informing concept	An example	Situation	
				`flow`	`stock`
`MassBased`	`Thermomech`	thermomech exergy [1]	penstock water	✓	✓
	`Chemical`	chemical exergy	natural gas	✓	✓
	`Total`	total exergy	*care required*	✓	✓
	`Material`	non-fuel uses	timber	✓	✓
	`Other`	other		✓	✓
`WorkTransfer`	`Electrical`	electrical work	AC power [2]	✓	—
	`Mechanical`	mechanical work	shaft power	✓	—
`HeatTransfer`	`Conduction`	heat	conduction [3]	✓	—
	`Radiation`	heat	thermal radiation	✓	—
`RecircMedia` [4]	`NetThermomech`	net-thermomech flow exergy	steam cycle	✓	—
	`NetChemical`	net-chemical flow exergy		✓	—
`Instrument`	`Funds`	finance	funds in [€]	✓	✓
	`Permit`	entitlement	Kyoto permits	✓	✓

Notes:
1. Thermomech is short for thermomechanical.
2. Uses a DC power flow representation.
3. Conduction is treated as a special case of logical flow.
4. Requires a pre-specified and interval-specific-invariant recirculation-rate.

Table 1: *Core taxonomy and class hierarchy for the C++ class* `Commodity` *which represents physical and instrumental resources (PIR).*

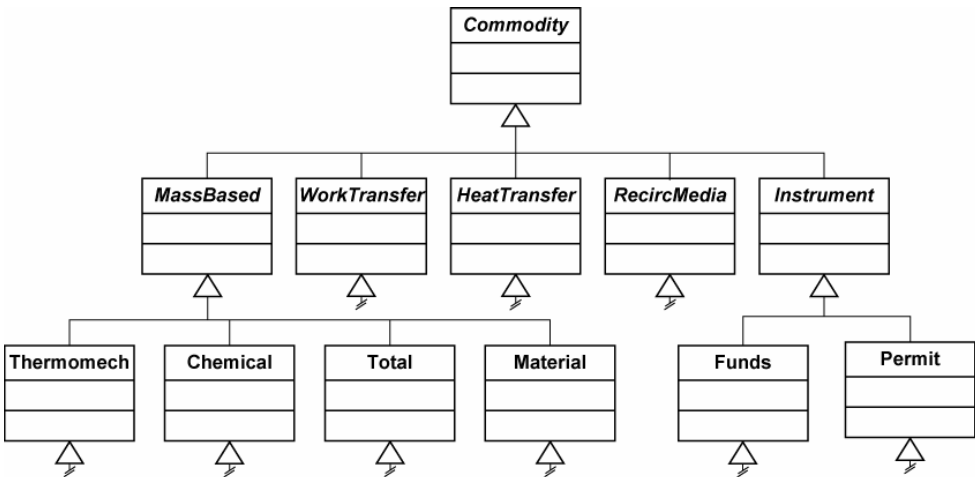

Figure 2: *A UML representation of the class hierarchy for physical and instrumental resources (PIR) given earlier in table 1.*

```
// class declarations (as opposed to definitions) with public inheritance

class MassBased  : public Commodity;      // matter-based PIR
class Chemical   : public MassBased;      // PIR valued for chemical exergy
class GasFuel    : public Chemical;       // gaseous oxidizable fuel

// object instantiation (constructor arguments indicated by ..)

GasFuel* maui_gas = new GasFuel( .. );    // from Maui field, New Zealand
GasFuel* biogas_a = new GasFuel( .. );    // from some gasification process
```

***Listing 1**: C++ pseudo-code covering class declarations and object instantiation for two gaseous fuel PIR. These two fuels would doubtless possess different attribute values, such as LHV (lower heating value) and density. And if not fundamentally substitutable, then another layer of inheritance could be established to enforce this restriction.*

5 Data design

Data design is concerned with data interchange and persistent storage. *xeona* uses a *single* XML file for each modeling project, rather than the filesystem through a collection of files and subdirectories. This same XML file is utilized for both input and output. New information can be added by either the *modeler* while preparing or developing a project or the *program* to store results at run-time. Hence, the central organizing concept is that of a *project*. Each project starts with a description of the common system under investigation — the *base system* — from which variants or *scenarios* are produced and run. The XML file also contains *meta-information* describing dataset attributes or indicating how particular model instances are to be evoked. The data design may exhibit some redundancy, although not sufficient to warrant concern. Of more importance is the fact that the hierarchical structure is designed so that each scenario is collected on discrete branches in order to reduce memory consumption (an XML file expands several-fold when held in memory in tree form). Further matters related to data design are noted in section 7. A project file (being text) can be maintained using the CVS utility, to enable roll-back and other version control features. Listing 2 shows an indicative XML dataset.

6 Software quality

Verification and validation are important aspects of software and model development. *Verification* refers to program correctness, whereas *validation* concerns model applicability in relation to the underlying assumptions. This section looks briefly at program correctness.

Software testing necessarily focuses on verification. The basic *unit of testing* is the (`.cc`) *source file* (and associated (`.h`) header file) rather than the class (Lakos 1996). That said, the act of preprocessing a source file prior to compilation automatically introduces new code *via* `#include` statements and related conditionals, to form a *translation unit*. Software test strategies focus on moving *up* the software component dependency graph in stepwise fashion and examining either total functionality thus far (*hierarchical testing*) or added functionality (*incremental testing*). The presence of complex dependencies, or worse, cyclic dependencies, can frustrate testing — one of the reasons why physical design needs to be approached with some care.

```xml
<?xml version="1.0" standalone="no"?>
<!-- information concerning structural compliance omitted -->
<xeona-dataset ver="0.00">
    <file-last-modified>
        <date time-zone="UTC">0000-00-00 00:00:00.00</date>
        <account>username@domain.name</account>
        ...
    </file-last-modified>
    <next-run-instructions id="1">
        ...
    </next-run-instructions>
    <project status="complete">
        <principal-author>Robbie Morrison</principal-author>
        <short-title>New Zealand national policy model</short-title>
        ...
    </project>
    <base-case status="complete">
        <time-step>
            <time unit="s">3600</time>
        </time-step>
        ...
        <entity id="1">
            <name>Huntly thermal power station</name>
            <xel>CV_ThermPowerStn_02</xel>
            <description> ... </description>
            <parameter_1 type="scalar" units="..."> ... </parameter_1>
            ...
        </entity>
        <connection id="1">
            ...
        <connection>
        ...
    </base-case>
    <scenario id="1" status="complete">
        <last-run>
            <account>usercode@domain.name</account>
            <xeona-version>0.00</xeona-version>
            <exit-status>success</exit-status>
            <start>
                <date time-zone="UTC">0000-00-00 00:00:00.00</date>
            </start>
            ...
        </last-run>
        <entity id="1">  <!-- details given in base case -->
            <reported-version>1.00</reported-version>
            <cost type="time-series" unit="EUR/s"> ... </cost>
            <co2e type="time-series" unit="kg/s"> ... </co2e>
            <coal type="time-series" unit="kg/s"> ... </coal>
            ...
        </entity>
        <entity id="86">  <!-- scenario-specific entry -->
            <name>Wellington wind farm</name>
            <xel>HA_WindFarm_01</xel>
            ...
            <cost type="time-series" unit="EUR/s"> ... </cost>
            ...
        </entity>
        ...
    </scenario>
    ...
</xeona-dataset>
```

***Listing 2**: An indicative XML dataset for* xeona *centered on the concept of a project and containing any number of scenarios. An actual dataset could easily run to many thousands of lines.*

A source file normally contains (among other things) one or more logical elements, normally class definitions. Testing a source file therefore means establishing that each class definition performs as expected. This, in turn, requires a *design* or *class specification* for each class. There is some tension between the concepts of checking classes and testing source files. One approach is to restrict each source file to a single class definition (plus any localized helper classes). Class specifications can then be used to progressively define *test regimes* for the various components that make up the program (as discussed earlier). Identifying test regimes should ideally form part of the software design process, a technique known as *design for testability* (DFT).

The second highest components in *xeona* comprise the core executable plus any dynamic libraries, and beyond these, the overarching script. Testing at the top level requires XML *test suites* based on representative project data. Test suites should contain both benchmark output and scripts which automate the data comparison process.

The related topic of software *exception handling* deserves brief mention. C++ supports `try`/`throw`/`catch` statements. However *xeona* makes modest uses of this construction, primarily because recovery options are normally limited. Furthermore, run-time resilience is not a design priority, whereas managing code complexity is. In most cases, it is simply better to display a message and either continue or exit as appropriate.

The use of open source software can also promote software quality — if only because code is written in the knowledge that it could well be subject to detailed public scrutiny.

7 Project extensions and outlook

The *xeona* model is being developed in stages. The heart of the project is the provision of core modeling functionality. Notwithstanding, model preparation, model interpretation, and data management are also key aspects, particularly from the *user viewpoint*. The adoption of a clean and extensible XML data model for data interchange is central to the development of these other features. In addition, a number of widely used data analysis tools now support XML and XSL (including Matlab and Excel).

Two areas are earmarked for development within this project. The first is *network visualization*, whereby the two key network structures within *xeona* are depicted in graphical form to aid usability. And the second is *web-based deployment* based on a client/server model — a feature which should make the program more attractive to some categories of user, although the popularity of Linux as a desktop replacement is steadily reducing this need.[19]

The question of extending the software design to take advantage of distributed computing systems is also under consideration for the obvious reason of numerical performance. However, no development strategies have been formulated at this point.

8 Concluding remarks

This paper reviewed aspects of the software design for *xeona* and sought to indicate and explain some of the more significant design decisions. The design process itself was divided into three realms for convenience: logical design, physical design, and data design. These realms clearly overlap, which means that a successful scheme will need to establish synergies and reconcile conflicts between each in a coordinated fashion.

Object-oriented programming is a powerful programming paradigm, well suited to systems modeling. XML supports an equally powerful data-centric paradigm for storing

[19] *deeco* is currently offered to authorized users though SSH2 secure shell and a command-line interface.

and manipulating hierarchically-organized data. Moreover, OOP and XML work well together. *xeona* builds on these strengths to create what should be a state-of-the-art policy support modeling environment, well matched to contemporary problems.

Notwithstanding, projects like *xeona* face a significantly higher design and coding overhead than perhaps a more traditional software approach would incur. This suggests that multi-group collaboration and an open source software (OSS) development model are probably essential for ongoing success.[20] The management of physically dispersed multi-developer projects introduces its own set of challenges. Still, a range of successful OSS projects indicate that an open and collaborative approach can work to advantage.

The OSS philosophy is also well suited to public policy support modeling. Because such models are invariably used in advocacy roles, the peer review process associated with OSS can help assure that the resulting conclusions are programmatically correct, methodologically defensible, and reasonable in context.

xeona is particularly suited to liberalized energy sector problems, because it is able to capture the *multi-agent dynamics* that exist in today's world, rather than posit (as many older models do) a single system agent making universally optimal choices. *xeona* is also able to support a *proactive approach* to policy analysis and can accommodate quite radical policy interventions, technical developments, and external contexts. Novel and as yet unproven technologies can, for instance, be tested "*in situ*" in circumstances where their engineering and financial performance is only approximately known. Statistical and/or behavior-centric modeling techniques, on the other hand, tend to perform less reliably when faced with circumstances which fall outside the data envelope within which they were developed and calibrated. Furthermore, *xeona* is able to capture fine-grain *network effects* first-hand. Such effects are only accessible to models with high temporal and topological resolution and can have a major bearing on the validity of modeled results, if such effects form important real-world dynamics.

The question of whether *xeona* will be applied to resource-based policy domains other than energy-services supply remains open at this juncture. There is every reason to believe that sustainability issues as diverse as water allocation, treatment, and reuse, or carbon sequestration and land-use choice, to name but two, would benefit from representative large-scale simulation.

Key abbreviations

CVS	concurrent versions system (a source code management utility)
deeco	dynamic energy, emissions, and cost optimization
DOM	document object model [XML]
DTD	document type definition [XML]
GCC	GNU compiler collection [compiler suite]
GLPK	GNU linear programming kit [component library]
GP	generic programming
GPL	GNU public license
GSL	GNU scientific library [component library]
MAS	multi-agent simulation
MILP	mixed integer linear program/programming

[20] These collaborations are being established progressively in the case of *deeco* and now *xeona*. Please contact the authors if interested in participating.

OOP	object-oriented programming (includes generic programming)
OSS	open source software
PIPC	public interest performance criteria [*xeona*]
PIR	physical or instrumental resource [*xeona*]
SAX	simple API for XML [XML]
UML	unified modeling language [OOP]
XEL	extensible entity library [*xeona*]
xeona	extensible entity-oriented optimization-based network-mediated analysis
XML	extensible modeling language [XML]
XSL	extensible stylesheet language [XML]

Acknowledgements

Robbie Morrison acknowledges support from New Zealand Government Bright Future Top Achiever Doctoral Scholarship TAD 523. Tobias Wittmann acknowledges support from the Stiftung der Deutschen Wirtschaft, Germany.

References

Barker, Terry. 1998. Large-scale energy-environment-economy modelling of the European Union. In Iain Begg and S.G.B. Henry (Eds.), *Applied economics and public policy*. Cambridge, UK: Cambridge University Press, 15–40. [ISBN 0-521-62414-2].

Beazley, David M., and Peter S. Lomdahl. 1997. *Building flexible large-scale scientific computing applications with scripting languages.* Paper presented at the Eighth SIAM Conference on Parallel Processing for Scientific Computing (PPSC 1997), Hyatt Regency Minneapolis on Nicollel Mall Hotel, Minneapolis, MN, USA, held 14–17 March 1997. [ISBN 0-89871-395-1].

Bejan, Adrian. 1997. *Advanced engineering thermodynamics — Second edition*. New York, NY, USA: John Wiley and Sons. [ISBN 0-471-14880-6].

Bejan, Adrian, George Tsatsaronis, and Michael J. Moran. 1996. *Thermal design and optimization*. New York, NY, USA: John Wiley and Sons. [ISBN 0-471-58467-3].

Berryhill, John B. 2001. *C++ scientific programming : computational recipes at a higher level.* New York, NY, USA: John Wiley and Sons. [ISBN 0-471-41210-4].

Bower, John, Derek W. Bunn, and Claus Wattendrup. 2001. A model-based analysis of strategic consolidation in the German electricity industry. *Energy Policy, 29*(12), 987–1005.

Breymann, Ulrich. 2000. *Designing components with the C++ STL : a new approach to programming — Revised edition*. Harlow, UK: Addison-Wesley. [ISBN 0-201-67488-2].

Bruckner, Thomas, Helmuth-M. Groscurth, and Reiner Kümmel. 1997. Competition and synergy between energy technologies in municipal energy systems. *Energy – The International Journal, 22*(10), 1005–1014.

Bruckner, Thomas, Robbie Morrison, Chris Handley, and Murray Patterson. 2003. High-resolution modeling of energy-services supply systems using *deeco* : overview and application to policy development. *Annals of Operations Research, 121*(1–4), 151–180.

Bruckner, Thomas, Robbie Morrison, and Tobias Wittmann. (*under review*). Public policy modeling of distributed energy technologies : strategies, attributes, and challenges. *Ecological Economics*.

Dattatri, Kayshav. 2002. *C++ : effective object oriented software construction : concepts, principles, industrial strategies and practices — Second edition*. Upper Saddle River, NJ, USA: Prentice Hall PTR. [ISBN: 0-13-086769-1].

Fowler, Martin, and Kendall Scott. 1999. *UML distilled : a brief guide to the standard object modeling language — Second edition*. Boston, MA, USA: Addison-Wesley. [ISBN: 0-2016-5783-X].

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design patterns : elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley. [ISBN 0-201-63361-2].

Gong, Mei. 2003. Optimization of industrial energy systems by incorporating feedback loops into the MIND method. *Energy – The International Journal, 28*(15), 1655–1669.

Gong, Mei, Magnus Karlsson, and Mats Söderström. 2002. *Industry and the energy market : optimal choice of measures using the MIND method.* Paper presented at the Power System and Communications Infrastructures for the Future, Beijing, China, held September 2002.

Groscurth, Helmuth-M., Thomas Bruckner, and Reiner Kümmel. 1995. Modeling of energy-services supply systems. *Energy – The International Journal, 20*(9), 941–958.

Groscurth, Helmuth-M., and K.-P. Kress. 1998. Fuzzy data compression for energy optimization models. *Energy – The International Journal, 23*(1), 1–9.

Illingworth, Valerie (Ed.) 1996. *Dictionary of computing — Fourth edition*. Oxford, UK: Oxford University Press. [ISBN 0-19-853855-3].

Jaeger, Carlo C., Marian Leimbach, Carlo Carraro, Klaus Hasselmann, Jean-Charles Hourcade, Andrew Keeler, and Rupert Klein. 1999. *Integrated assessment modeling : modules for cooperation*. Potsdam, Germany: Potsdam Institute of Climate Impact Research.

Kainuma, Mikiko, Yuzuru Matsuoka, and Tsuneyuki Morita. 2003. *Climate policy assessment : Asia-Pacific Integrated Modeling*. Tokyo, Japan: Springer Verlag. [ISBN: 4-431-70264-4].

Lakos, John. 1996. *Large-scale C++ software design*. Boston, USA: Addison-Wesley. [ISBN 0-210-63362-0].

Lee, Richard C., and William M. Tepfenhart. 2000. *UML and C++ : a practical guide to object-oriented development — Second edition*. Upper Saddle River, NJ, USA: Prentice Hall PTR. [ISBN 0-13-029040-8].

Lischner, Ray. 2003. *C++ in a nutshell : a language and library reference*. Sebastopol, CA, USA: O'Reilly and Associates. [ISBN 0-596-00298-X].

Loudon, Kyle. 2003. *C++ pocket reference*. Sebastopol, CA, USA: O'Reilly and Associates. [ISBN 0-596-00496-6].

Morrison, Robbie, and Thomas Bruckner. 2002. *High-resolution modeling of distributed energy resources using deeco : adverse interactions and potential policy conflicts.* Paper presented at the 3rd International Workshop : Advances in Energy Studies : Reconsidering the Importance of Energy, Porto Venere, Italy, held 24–28 September 2002.

Morrison, Robbie, Tobias Wittmann, and Thomas Bruckner. 2003. *Energy policy and distributed solutions : a model-based interpretation.* Paper presented at the Australia New Zealand Society for Ecological Economics (ANZSEE) Think Tank, University of Auckland, Auckland, New Zealand, held 16 November 2003.

Noth, Michael, Alan Borning, and Paul Waddell. 2003. An extensible, modular architecture for simulating urban development, transportation, and environmental impacts. *Computers, Environment and Urban Systems, 27*(2), 181–203.

Ousterhout, John K. 1998. Scripting : higher-level programming for the 21st century. *IEEE Computer,* (March), 23–30.

Outhred, Hugh R., and R. John Kaye. 1996. Incorporating network effects in a competitive electricity industry : an Australian perspective. In Michael A. Einhorn and Riaz Siddiqui (Eds.), *Electricity transmission pricing and technology*. Boston, Massachusetts, USA: Kluwer Academic Publishers, 207–228. [ISBN 0-7923-9643-X].

Philpott, Andrew B. 1999. *Experiments with load flow pricing models.* Paper presented at the CRNEC Policy Conference, University of Auckland, Auckland, New Zealand, held 01–02 September 1999.

Pilone, Dan. 2003. *UML pocket reference*. Sebastopol, CA, USA: O'Reilly and Associates. [ISBN 0-596-00497-4].

Remme, Uwe, Gary A. Goldstein, Ulrich Schellmann, and Christoph Schlenzig. 2001. *MESAP/TIMES : advanced decision support for energy and environmental planning.* Paper presented at the International Conference on Operations Research (OR 2001), Duisburg, Germany, held 03–05 September 2001. [ISBN 3-540-43344-9].

Riahi, Keywan, and R. Alexander Roehrl. 2000. Robust energy technology strategies for the 21st century : carbon dioxide mitigation and sustainable development. *Environmental Economics and Policy Studies, 3*(2), 89–123.

Robbins, Kay A., and Steven Robbins. 2003. *Unix systems programming : communication, concurrency, and threads — Second edition*. Upper Saddle River, New Jersey, USA: Prentice Hall PTR. [ISBN 0-13-042411-0].

Roussel, Catharine, and Marc R. Roussel. 2003. Generic object-oriented differential equation integrators. *C/C++ Users Journal, 21*(11), 18–23.

Sedgewick, Robert. 2002. *Algorithms in C++ — Third edition / Part five : Graph algorithms*. Boston, USA: Addison-Welsey. [ISBN 0-201-36118-3].

Shang, Zhigang, and Antonis C. Kokossis. 2002. *On the integration of thermodynamics with mathematical programming for total site optimization.* Paper presented at the ECOS 2002 : 15th International Conference on Efficiency, Costs, Optimization, Simulation, and Environmental Impact of Energy Systems, Berlin, Germany, held 03–05 July 2002. [ISBN 3-00-009533-0].

Siek, Jeremy G., Lie-Quan Lee, and Andrew Lumsdiane. 2002. *The Boost graph library : user guide and reference manual*. Boston, MA, USA: Addison-Wesley. [ISBN 0-201-72914-8].

Sycara, Katia. 2003. *Multi-agent systems*. From http://www.aaai.org/AITopics/html/multi.html.

Tomson, Phil. 2004. Mixed-language development and SWIG. *C/C++ Users Journal, 22*(1), 6–10.

Weinhardt, Christof, Pascal Zuber, Mathias Göbelt, Wolf Fichtner, Martin Wietschel, and Otto Rentz. 2000. *A competition-model for the electricity sector : from an OR approach to a multi-agent system.* Paper presented at the Workshop 2000 : Agent-Based Simulation, Ghent, Belgium, held 02–03 May. [ISBN 1-56555-185-0]. ❑