EChem++−An Object-Oriented Problem Solving Environment for Electrochemistry. 2. The Kinetic Facilities of Ecco−A Compiler for (Electro-)Chemistry†

Kai Ludwig and Bernd Speiser*

Institut für Organische Chemie, Universität Tübingen, Auf der Morgenstelle 18, D-72076 Tübingen, Germany

Received July 13, 2004

We describe a modeling software component Ecco, implemented in the C++ programming language. It assists in the formulation of physicochemical systems including, in particular, electrochemical processes within general geometries. Ecco's kinetic part then translates any user defined reaction mechanism into an object-oriented representation and generates the according mathematical model equations. The input language, its grammar, the object-oriented design of Ecco, based on design patterns, and its integration into the open source software project EChem++ are discussed. Application Strategies are given.

1. INTRODUCTION

The solution of electrochemical research problems conventionally relies on numerical simulation models, which are implemented in the form of computer programs.[2] State-of-the-art models of electrochemical systems take into account homogeneous chemical reactions coupled to electron transfer, adsorption, and other processes. A manifold of experimental techniques within various, even complex cell geometries have been treated.[3−7] However, additional steps or process types may prove important in future simulation studies and will have to be included in the calculations.

In view of this development, such modeling and simulation software demands for a programming methodology that both copes with high logical complexity and offers an easy way to reuse and to extend components for future problems: object-orientation is a highly promising concept for that purpose.

On this basis, recently, the development of EChem++, a problem solving environment (PSE[8]) for electrochemistry, was initiated.[9] Its framework includes modeling and simulation tasks and real time control of electrochemical experiments as well as data analysis (for a brief view of the overall design of EChem++, see refs 1 and 9).

Herein, we continue the discussion of this PSE and its implementation with another crucial component: Ecco (**E**lectro **C**hemical **CO**mpiler), a modeling tool for electrochemical experiments. We will use the term *modeling* here to designate the process of formulating and defining the details of a particular physicochemical model of real systems and processes. The subsequent task of solving the governing mathematical equations by means of numerical methods will be called *simulation*. Ecco includes concepts for handling almost arbitrary chemical reactions as well as general geometries. A precise specification of geometric aspects, however, will be deferred to future work.

The kinetic facilities of Ecco described here deal with the translation of a user defined reaction mechanism (*reaction network*) into the underlying kinetic expressions of the governing equations, i.e., the reaction terms added to variants of nonstationary Nernst−Planck equations[10] and the source and sink terms applied in boundary conditions.

Such *kinetic compilers* (synonyms: *reaction compilers* and *chemical compilers*) are well-known in chemical reaction kinetics (see refs 11−13 and references therein). However, compilers concerning the kinetics of chemical reactions coupled to heterogeneous electron transfers are less frequent. Two of them are included in the commercial simulation package DigiSim[14] and in the ELSIM system.[15] Several similar software packages exist.[16,17] For the present discussion, we choose these two particularly popular and representative examples.

The compiler included in DigiSim provides a chemically intuitive input language, which is appropiate for first- and second-order homogeneous chemical reactions and heterogeneous electron transfers. The latter are formulated as single electron reductions only. Presuming correct *reaction stoichiometry* [The term *reaction stoichiometry* denotes the stoichiometric relation between reacting species and neglects their elementary (atomic) composition.] in the formulated input mechanism, *thermodynamically superfluous reactions* (TSRs) are determined, and consistent parameters are calculated.[18] However, the compiler does not detect errors in reaction stoichiometry within the reaction network. For homogeneous reactions, the reaction order of a reacting species is assumed to be equal to its stoichiometric coefficient, which is not necessarily true.

As long as only a rather limited kinetical spectrum is considered, the DigiSim compiler stands out as an easy to use component. However, more complex kinetic processes, arising in e.g. catalysis, can only be simulated by formulating a series of elementary steps. Neither adsorption processes nor reactions in 2- or 3-dimensional geometries are considered. Among several other restrictions (see ref 13 for a comprehensive discussion), the commercial character of DigiSim prevents the reuse and extension of its compiler. As a work-around, for example, recently, a "trick" employing obviously nonphysical process steps had to be used in order to extend the ability of DigiSim to simulate electrogenerated chemiluminescence.[19] Furthermore, owing to the nonavail-

* Corresponding author e-mail: bernd.speiser@uni-tuebingen.de.

† Part 1: see ref 1.

10.1021/ci0497814 CCC: $27.50
Published on Web 10/09/2004

ability of the source code, an objective verification of the internal translation procedure is not possible. Most likely, judging from appearance, a compiler based on similar principles is included in DigiElch.[16]

ELSIM's compiler, which is written in the C programming language, on the other hand, is described in detail in a number of papers.[13,15,20,21] It copes with a wide range of electrochemical models, including equilibrium adsorption processes. A check for both violation of mass balance (incorrect reaction stoichiometry) and presence of TSRs is performed. The kinetics of homogeneous chemical reactions are a priori assumed to obey a power rate law, thus precluding more complex mathematical relationships.

ELSIM's compilation process is composed of two parts. First, a *reaction compiler* performs the translation of the user defined reaction mechanism into a *target text* of the governing equations. Human readable *problem description files* are created to give a "proper understanding of the underlying mathematical formalism".[13] The target text can be verified and modified by the user. Thus, assuming sufficient mathematical knowledge, more complex kinetic models can also be simulated. For technical reasons, the total number of species within the reaction mechanism is limited to 20.[15] In a second step, a *formula translator* acts as an *interpreter* that generates the internal simulation algorithm. Therefore, the performance of the simulation depends to some extent on the efficiency of the formula interpretation.[15]

Compared to DigiSim, the input language of the reaction compiler appears to be considerably more complex.[13] As with DigiSim, higher dimensioned geometric models are not considered. The source code is closed.[22] Therefore, a comprehensive discussion of the software design and thus the extendability of the compiler is impossible. However, from the use of the C programming language, which does not directly support an object-oriented design, we have to assume that a relatively high effort would be necessary to enrich or modify the compiler in the context of future developments.

To allow the implementation of an improved, flexible, and extendable electrochemical PSE, some of the problems discussed above will be tackled within Ecco.

The overall requirements which we impose on the kinetic facilities of Ecco include the support of heterogeneous electron transfer reactions, adsorption processes, and homogeneous reactions (occurring in a gas phase, in solution, or on a surface) combined with a nonrestricted variety of kinetic laws. The input language should be defined as simply as possible and consists of chemically intuitive elements. It should include a straightforward concept to model (electro)-chemical reactions within general geometries. Mass balance within the reaction network should be checked. Then, in the context of TSRs, consistent sets of thermodynamic parameters may be automatically determined.

For the implementation of the compiler we will discuss an object-oriented design that both ensures simple reusability within other software and provides a flexible concept for future extensions. The efficiency of the actual simulation will not be influenced by a formula translation step. It is important to note that no specific assumptions about the discretization methods used for the simulation of the mathematical model will be made.

Table 1: Components of Reaction Networks Classifiable by Ecco.

component	chemical notation (example)	representation[a]	syntax rule[b]
dissolved, neutral species	A	A	neutrum
adsorbed, neutral species	A_{ads}	A{ads}	adsNeutrum
dissolved, charged species	A^{1+}	A^(1+)	ion
adsorbed, charged species	$A^{1+}{}_{ads}$	A^(1+){ads}	adsIon
electron	e^-	e-	electron
nonequilibrium reversible reaction	$\rightleftharpoons$	$\leftrightarrow$	rev
equilibrium reaction	$\rightleftharpoons$	=	equil
irreversible reaction	$\rightarrow$	$\rightarrow$	irrev

[a] Input language of Ecco. [b] See grammar definition (section 2.2).

First, we will describe the basic concepts used and the processes classified by Ecco in its present version as well as its input language (section 2.1). The grammar is defined in section 2.2. In section 2.3 we discuss the object-oriented design of the compiler as well as algorithmic aspects, while application strategies are given in section 2.4.

2. RESULTS AND DISCUSSION

2.1. Object-Oriented Analysis and Input Language. (Electro)chemical reaction networks to be classified by Ecco represent a sequence of reactions which in turn consist of components (Table 1) that describe reacting entities and their relation (reaction type). Several pieces of kinetic and geometric information can be associated with these reactions and may be specified as options.

The current version of Ecco's kinetic compiler distinguishes between two basic types of entities that occur in electrochemical reactions: electrons and chemical species. While the species name e- is reserved for the electron, the name of a chemical species can be defined almost arbitrarily. Chemical species can be further classified into dissolved species, present in solution or in a gas phase, and adsorbed molecules, which are immobilized on a surface. Both forms can be charged, which possibly influences their transport behavior during an electrochemical experiment.

Homogeneous reactions are classified as reactions occurring exclusively between dissolved species, e.g. eq 1. Reactions between adsorbed species are termed surface reactions, e.g. Reaction 2. Note that in Ecco's input language, each reaction line is closed by a semicolon (section 2.2). Throughout this section, we contrast examples for some usual chemical notation, in Roman font, with the corresponding input language formulation of Ecco, in typewriter font.

$$3A + B \rightleftharpoons 2C + D$$

$$\texttt{3A + B <=> 2C + D;} \qquad (1)$$

$$3A_{ads} + B_{ads} \rightleftharpoons 2C_{ads} + D_{ads}$$

$$\texttt{3A\{ads\} + B\{ads\} <=> 2 C\{ads\} + D\{ads\};} \qquad (2)$$

Reaction types are characterized by various reaction arrows (Table 1). Nonequilibrium reversible reactions are described by a forward and a backward rate constant, k_f and k_b, with finite numerical values. For high values of k_f and k_b, the reaction converges to an equilibrium state, and the concen-

trations of the reacting species are described by an algebraic relationship, the law of mass action. On the other hand, reactions with $k_b = 0$ are irreversible. Obviously, both equilibrium and irreversible reactions are special cases derived from the nonequilibrium reversible case, which is used for the examples in this section.

For simple systems, the kinetics of homogeneous reactions are commonly assumed to obey a power rate expression[23]

$$\frac{1}{\nu_k}\partial_t u_k = k_f \prod u_i^{\alpha_i} - k_b \prod u_j^{\beta_j} \tag{3}$$

where ∂_t denotes $\partial/\partial t$, ν_k is the stoichiometric coefficient of a species with index k and dimensionless concentration $u_k = c_k/c_{ref}$, where the reference concentration, c_{ref}, is usually the initial concentration of an educt. We have $\nu_k < 0$ for educts (i) and $\nu_k > 0$ for products (j).

As the other electrochemical compilers mentioned, Ecco by default assumes that the exponents α_i and β_i equal the absolute values of the stoichiometric coefficients. In general, however, this is not always consistent with experimental observations. We thus provide a mechanism to change α_i and β_i to any real number by adding an option to the according reaction, as long as the relation $\nu_i = (\beta_i - \alpha_i)/n$ is fulfilled for species i and any real number $n > 0$. This relation preserves equilibrium conditions, as described in ref 23. For example, the rhodium catalyzed formation of methanol may be written as[23]

```
2 CO + 3 H2 <=> CH3OH + CO + H2; p<CO> =
                (-0.3, -1.3), p<H2> = (1.3, -0.7);  (4)
```

According to the options formulated after the first semicolon, the powers of the species CO and H2 assume the values $(\alpha,\beta)_{CO} = (-0.3, -1.3)$ and $(\alpha, \beta)_{H_2} = (1.3, -0.7)$, respectively, and are used in the power rate laws.

Complex reactions, e.g. chain or catalytic reactions,[23] sometimes obey nonpower rate laws. Therefore, Ecco supports *generic rates*, $(1/\nu_k)\partial_t u_k = r(u_i,...)$, definable as arbitrary algebraic expressions composed of species concentrations occurring within the reaction network, user defined constants and a set of standard mathematical functions. Such terms are written after the corresponding chemical reaction. For example, Michaelis−Menten kinetics can be formulated as

$$S + E \overset{K_0}{\rightleftharpoons} ES$$

$$ES \overset{k_{f1}}{\rightleftharpoons} P + E \qquad r = \frac{k_{f1}[E]_0[S]}{K_0 + [S]}$$

```
S + E <=> ES;

ES <=> P + E; r = kf1 * E0 * c<S>/(K0 + c<S>);  (5)
```

where K0 denotes the classical equilibrium constant of the first reaction and kf1 the forward rate constant of the second reaction. Note, that the kinetics of the second reaction no longer follow the standard power rate law as shown in eq 3, and, consequently, this example shows that formulations in the context of Ecco are not restricted to this special case.

Adsorption/desorption processes occur as interaction between molecules in the vicinity of a surface (s) and the surface itself:

$$A{\cdot}s \rightleftharpoons A_{ads}$$

```
A <=> A{ads};  (6)
```

Frequently, the adsorption step is assumed to be unimolecular and at equilibrium. Therefore, equilibrium adsorption isotherms have been applied most often to simulate adsorption processes in electrochemical contexts.[24−29]

Ecco, however, by default, supports the dynamic model of Langmuir including coadsorption of several species[23](pp 192−194):

$$\partial_t \theta_k = k_f[1 - \sum_j \theta_j]{\cdot}u_k - k_b{\cdot}\theta_k \tag{7}$$

$\theta_k = \Gamma_k/\Gamma_s$ denotes the normalized surface concentration (fractional coverage) of the adsorbed form of species k with normalized solution concentration u_k, real surface concentration Γ_k, and saturation concentration Γ_s. The summation term is due to the occurrence of coadsorbed species on the same surface. Here, k_f and k_b are the adsorption/desorption rate constants.

Other adsorption isotherms (i.e. Henry, Frumkin, and Temkin isotherms) are provided through optional keywords.

Heterogeneous electron-transfer reactions (formulated either as reduction or as oxidation; eq 8) occur at electrode surfaces and involve dissolved species transported from/into the adjacent solutions or adsorbed species located on the surface. Ecco identifies such reactions by the presence of the e- symbol. The involved species are termed electro- or redox-active.

$$A + e^- \rightleftharpoons B$$

$$C_{ads} \rightleftharpoons D_{ads} + 2e^-$$

```
A + e- <=> B;

C{ads} <=> D{ads} + 2 e-;  (8)
```

The kinetic model of Butler−Volmer, formulated for a reduction, is currently supported:

$$\frac{1}{\nu_k} D_k \nabla_n u_k = -\psi[\exp(-\alpha p(t)){\cdot}u_k - \exp((1-\alpha)p(t)){\cdot}u_j] \tag{9}$$

In eq 9, $p(t) = (nF/RT)[E(t) - E^\circ]$, where $E(t)$ denotes the time dependent potential (including the special case of constant E), $\nabla_n u_k$ is the gradient of u_k normal to the electrode surface, D_k is the diffusion coefficient, α is the transfer coefficient, and ψ is the (dimensionless) standard heterogeneous rate constant of the electron transfer. The other symbols have their usual meanings. In the case of a potential controlled experiment, $p(t)$ corresponds to the potential excitation function, and eq 9 can be used in the calculation of the boundary value variation at an electrode surface.

The examples discussed above are sufficient to define various electrochemical mechanisms within a one-dimensional model geometry in which all the heterogeneous steps or surface reactions occur at the same boundary. However,

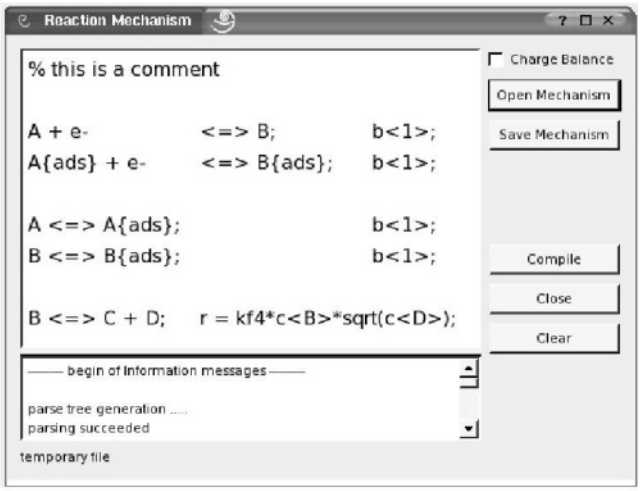

Figure 1. Example of an electrochemical reaction mechanism, formulated in Ecco's input language. The kinetics specified as a generic rate follow an example described in ref 23, p 188; **kf4** is the forward rate constant of reaction 4, i.e. B ⇌ C + D.

more general geometric assumptions demand for an additional concept. An inspiring notation arises from a common mathematical formulation of boundary value problems, whereby the domain under study is framed by numbered boundary segments.[7] Therefore, we introduce *boundary indices* (b⟨*n*⟩, *n* integer), that have to be assigned to all reactions affected by a boundary:

$$A \cdot s \rightleftharpoons A_{ads} \text{ on boundary } 1$$

$$A \Leftrightarrow A\{ads\};\ b\langle 1\rangle; \qquad (10)$$

This concept enables a flexible reaction modeling in one, two, and even three spatial dimensions.[7] It offers the possibility to interlink the kinetic part of Ecco with existing (see e.g. ref 30) or future computer aided design software.

Figure 1 shows the formulation of an example electrochemical reaction mechanism in Ecco's input language. It combines adsorption steps of two species A and B, electron transfer of redox-active A and a follow-up reaction of the electro-generated product B. All heterogeneous steps occur on a boundary indexed by 1.

Implications of the concepts proposed here to electrochemical research will be discussed separately.[31]

2.2. Grammar. The input to Ecco, i.e., the sequence of (electro)chemical reactions with options, can be regarded as a programming language adapted to the requirements to formulate electrochemical models. The core of each programming language is the definition of its grammar. Generally, a grammar is based on a *start symbol* defined by one or more *syntax rules*. Each of these syntax rules can again be defined by a *sequence* of syntax rules, which leads to *subsequences* that possibly include recursive constructs. The irreducible elements of a language are called *terminal symbols*. The terminology used here is based on the *Extended Backus-Naur Form* (EBNF).[32]

Figure 2 shows an excerpt from Ecco's grammar definition. The notation follows the EBNF-like formalism provided by the Spirit library.[33] The starting symbol is the reactionNetwork which is defined by one or more syntax rules termed reactionBlocks. Each of the reactionBlocks consists of a reaction closed by the terminal symbol ';'

```
reactionNetwork
  = reactionBlock >> *( reactionBlock );

reactionBlock
  = reaction >> ';' >> !(options);

reaction
  = educts >> reactionArrows >> products;

reactionArrows
  = rev | irrev | equil;

educts
  = reactionAddend >> *( '+' >> reactionAddend );

products
  = reactionAddend >> *( '+' >> reactionAddend );

reactionAddend
  = !(coefficient) >> species;

species
  = electron | adsIon | adsNeutrum | ion | neutrum;

neutrum
  = +( alnum | '(' | '[' ) >> *( alnum | ']' | ')' | '_' | '-');

options
  = ( boundaryOperator >> !( ',' >> laws ) >> ';' )
  | ( laws >> !( ',' >> boundaryOperator ) >> ';' );

laws
  = adsorptionLaw | electrodeLaw | homogeneousLaw;

homogeneousLaw
  = powerLaw | genericRate;

genericRate
  = 'r' >> '=' >> expression;
```

Figure 2. Grammar excerpt of Ecco's input language. Each line starts with the name of a nonterminal symbol followed by its definition. The symbol '≫' means *followed by*, '*' *zero or more*, '+' *one or more*, '!' *zero or one*, and '|' denotes *or*. Terminal symbols are enclosed by quotation marks. The alnum symbol denotes an alphabetic character or a number.

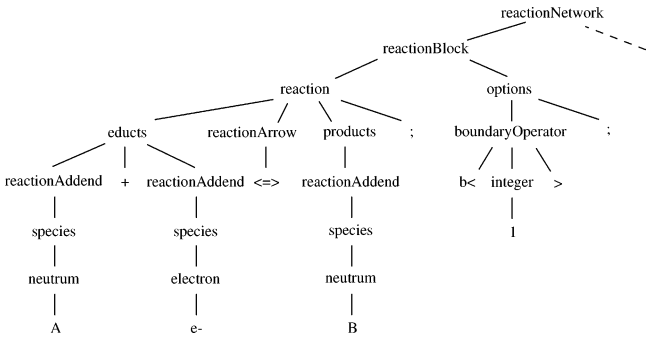

Figure 3. Ecco parse tree excerpt. The shown branch represents the first reaction of the example given in Figure 1.

and is optionally followed by options. The definitions of the subsequences that define a chemical reaction are straightforward.

The options consist of a boundaryOperator and/or laws. As for reactions, each option section has to be closed by a semicolon.

Beside the rules defining the type of adsorption (adsorptionLaw), heterogeneous electron transfer (electrodeLaw), or power rate laws (powerLaw), somewhat more complex subsequences are specified for generic rate laws (genericRate). As shown by the example in Figure 1, the compiler recognizes a generic rate as a sequence opened by the 'r' and '=' terminals, followed by an algebraic expression. A parse tree excerpt derived for the example mechanism of Figure 1 is shown in Figure 3. The compiler's algorithm is based on the processing of such parse trees.

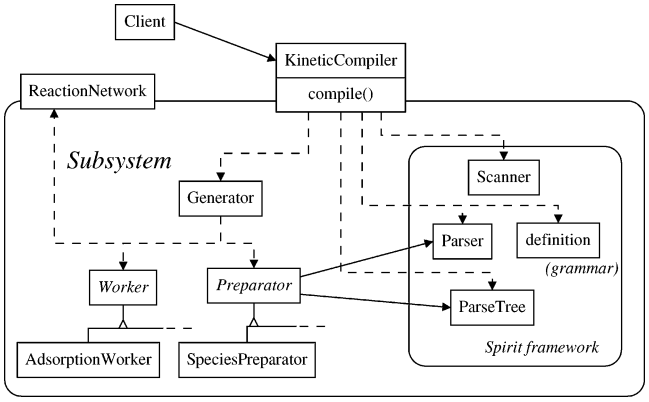

Figure 4. Overall object-oriented design of Ecco's kinetic compiler. The outer frame encloses the classes used during the translation process (subsystem). The inner frame embraces the classes needed or created within the Spirit parser generator framework (see text).

2.3. Object-Oriented Design and Algorithm. The discussion of the kinetic compiler is divided into two logical blocks. First, in sections 2.3.1 and 2.3.2, we describe the well extendable design of the compiler itself (Figure 4), in particular that of its translation process and discuss algorithmic aspects. Second, in section 2.3.3, we focus on the structure of the compiler's output which provides a concept to assemble discretized model equations later on.

Several design patterns are applied, i.e., the *Facade*, *Command*, *Proxy*, *Composite*, *Mediator* patterns,[34] as well as a generic form of the *Observer* pattern.[35] The symbols used in the diagrams follow the notation explained in ref 34.

2.3.1. The Kinetic Compiler. Conceptually, a compiler is build upon several *phases* that can be collected into two *parts*.[36] For example, *lexical analysis*, *syntax analysis*, and *semantic analysis* are assumed as phases of the *analysis part*, while *intermediate code generation* and *code generation* form the *synthesis part*. All phases are accompanied by error detection and data management. Ecco's KineticCompiler is an implementation of the facade pattern (Figure 4). It represents a facade to a subsystem that includes a manifold of different interacting objects. The facade provides a compile() function that delegates all the translation work to the subsystem and, finally, returns a ReactionNetwork object as the compiler's output.

For the analysis part, *scanner generators* and *parser generators* have been developed.[36−41] In particular, the Spirit library implements an object-oriented recursive-descent parser generator framework.[37] In contrast to other parser generators, Spirit does not require additional external compilation procedures and can be embedded completely within C++ sources. This should provide considerable advantages in terms of efficiency and clarity of the code, and, consequently, Spirit was employed for the analysis part of Ecco's kinetic compiler (inner frame in Figure 4). Thus, the KineticCompiler builds a ParseTree object (see also section 2.2), according to the grammar specification of the input language (defined in a definition class nested inside the Parser). The ParseTree object represents the user defined input mechanism. A Scanner class, internal to the Spirit library and compatible with iterators supplied by C++ standard template library (STL)[33,42] containers, analyzes the input. Syntax errors are detected, and the appropriate exception handling is performed.

Within the synthesis part of the compilation, a Generator object controls the algorithm that builds an object-oriented representation of the reaction mechanism and assembles reaction and boundary terms for each species. The Generator further structures this process by employing *preparators* and *workers* (*synthesis objects*).

In a first step, Preparator objects traverse the ParseTree and store transient information about basic components and options, such as rate laws and boundaries. Matrices that represent the reaction stoichiometry[18,20] are generated:

$$\sum_k e_{ik} \cdot A_k \rightleftharpoons \sum_k p_{ik} \cdot A_k \tag{11}$$

$$E = (e_{ik})_{i,k},\ P = (p_{ik})_{i,k} \tag{12}$$

$$S = (\nu_{ik})_{i,k} = P - E \tag{13}$$

where $i = 0, ..., N_r - 1$, $k = 0, ..., N_s - 1$. Symbols N_r and N_s denote the total number of reactions and species A_k, respectively. For reaction i (eq 11), E and P (eq 12) store the stoichiometric coefficients of educts (e_{ik}) and products (p_{ik}), respectively. Furthermore, the overall "stoichiometric matrix" S (eq 13) is calculated. Several persistent objects, such as species and generic rate law objects, are created and sorted by their types. Figure 5 shows a more detailed view of the command pattern which implements this first synthesis step. The generator aggregates several preparators. The abstract Preparator class prescribes a function traverse() to all the derived preparator classes. Data management is realized in TransientData and NetworkData objects (Figures 5 and 6).

A similar design is applied to the second synthesis step of the compiler. Based on the stoichiometric matrices and other information collected by the preparators, the Workers provide an execute() operation to create objects representing chemical reactions and the according rate laws. Reaction and boundary terms are assembled.

In both synthesis steps several consistency checks are performed (see section 2.3.2).

Figure 6 shows the interaction diagram of the synthesis part described above.

At the construction time (instantiation) of the generator, the sequence of the algorithm is defined, i.e., the synthesis objects are instantiated, and stored such that the traverse() and execute() functions are called successively as requested by logical dependencies between the preparators and/or workers. Simultaneously, the synthesis objects get pointers to the data structures of the two data objects (solid arrows from right to left). Then, the generator starts the two synthesis steps. Note that after the destruction of the generator only the network data and the reaction network object remain instantiated. At present, the complete synthesis part includes altogether 10 preparators for the first and 10 workers for the second step.

Although the whole synthesis process relies on an enhanced use of the memory extensive associative STL containers *map* and *set*, efficiency is preserved in two ways: first, the (empty) data structures are created only once within either the TransientData or the NetworkData objects.

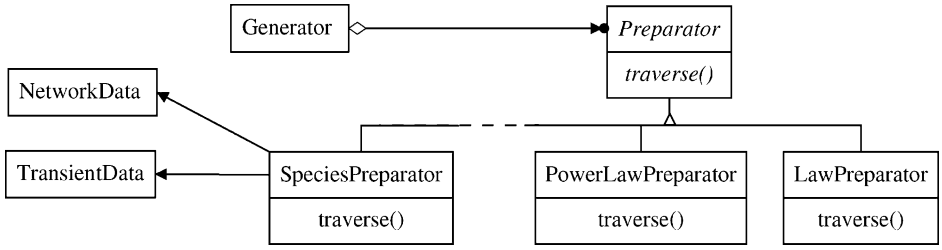

Figure 5. Command pattern of the first synthesis step. The hierarchy shows selected Preparator classes.

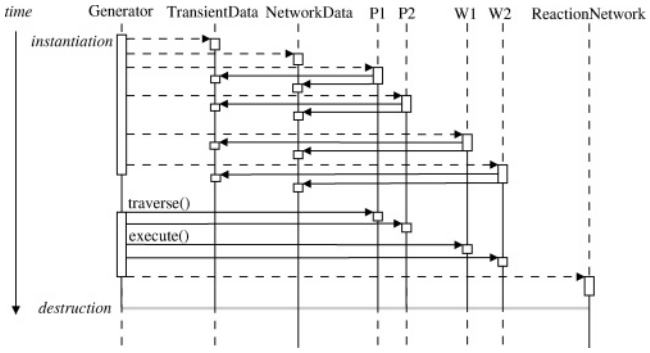

Figure 6. Interaction diagram of the synthesis part starting from the instantiation of the generator. A vertical solid line denotes the lifetime of an instantiated object, and boxes symbolize active operations on that object. Dashed arrows describe that a new object is created, and solid arrows denote requests to an object. P1, P2, W1, and W2 represent examples for **p**reparators and **w**orkers, respectively.

Second, both objects provide full read and write access to their data through pointers.

2.3.2. Consistency Checks. Beside the detection of species, reactions, and rate laws, as well as the creation of appropiate objects, Ecco checks the correctness of the user defined reaction mechanism in various ways.

Once the stoichiometric matrix S has been established by a preparator according to eq 13, the compiler checks stoichiometric consistency (mass balance).[12,20] Mathematically, mass balance within a reaction mechanism can be expressed as

$$S \cdot m = 0 \tag{14}$$

where $m \in \mathbb{R}^{N_s \times 1}$, $m > 0$ denotes a vector of necessarily positive masses (entries m_k of m) of all species involved in the reactions at equilibrium. There exists a $\lambda > 0$ such that

$$S \cdot y = 0 \tag{15}$$

holds for $y = \lambda \cdot m$, where $y \geq 1$. With $x = y - 1$ and without loss of generality concerning the objective function $z = -\sum_k x_k + 1$, mass balance is equivalent to a valid solution of the linear programming problem

$$\text{maximize } z = -\sum_k x_k + 1, \text{ with } S \cdot x = b,\ x \geq 0 \tag{16}$$

and $b = -S \cdot \mathbf{1}$, where vector $\mathbf{1}$ contains only unity elements. Problem (16) is solved by a BalanceWorker object that controls an implementation of the Simplex algorithm.[43]

Optionally, as shown in Figure 1 (top right-hand corner of window), Ecco checks charge balance within each reaction.

Additionally, Ecco includes a mechanism to check parameter consistency in contexts of cell geometries with multiple electrode arrangements and generic rate laws as well as TSRs.[18]

Suppose electron-transfer reactions occur at N_e different electrode immersed in the same solution. A typical example might be a ring-disk electrode.[44] Then, for each redox couple, Ecco creates N_e ETReaction objects supplied with rate laws and reaction parameters, since the kinetic parameters as well as the excitation signals applied at the electrodes may be different. However, the values of the formal potentials E^0_e ($e = 0, ..., N_e - 1$) of the electron transfers must be identical, as E^0 is assumed to be independent of electrode properties. Therefore, if a E^0_j value is changed by the user for any $j \in [0, N_e - 1]$, it has to be guaranteed that each of the other E^0_e values with $e \neq j$ obtain the same value.

A similar situation arises if several generic rate laws contain a parameter that belongs to another reaction within the mechanism, e.g. K0 in eq 5. In the grammar definition of generic rates, the symbols kf, kb, and K are reserved keywords. Combined with the index of the according reaction, grammar rules such as equilibriumConstant = 'K' $\gg$ +digit are defined, where digit denotes a number between 0 and 9 (for the meaning of the other symbols see Figure 2). If present within a generic rate, such constructs are detected and thus represented as nodes within the parse tree. Then, a preparator object determines sets of interdependent parameters, as with the E^0_e values described above.

In the context of TSRs, the relation between equilibrium constants of homogeneous and electron-transfer reactions is given by[18,20,45]

$$K_r^{\omega_r} = \prod_{j=0, j \neq r}^{N_{tsr}-1} K_j^{\omega_j} \tag{17}$$

where N_{tsr} denotes the number of "dependent reactions".[20] The K values for electron transfers are related to the formal potentials E^0 as[20]

$$K = \exp\left[-\frac{\nu_e F}{RT} E^0\right] \tag{18}$$

with the stoichiometric coefficient of the electron ν_e, the Faraday constant F, the gas constant R, and the temperature T. For dependent reactions the stoichiometric vector $s_r = (\nu_{rk})_k$ (row of the stoichiometry matrix S in eq 13, except for a change in indexing) can be expressed by a linear

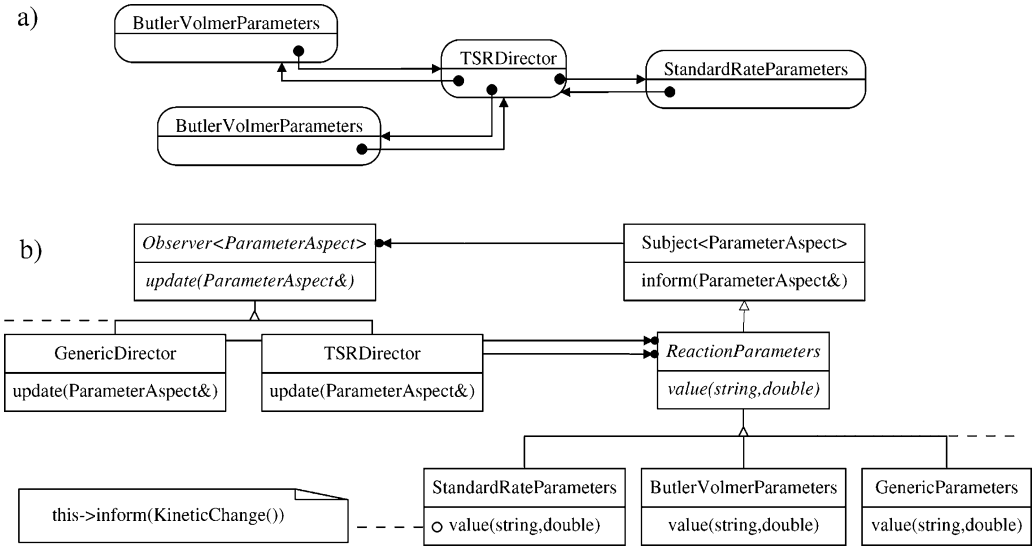

Figure 7. Example object structure in case of TSRs (a) and according class hierarchy (b) that combines forms of the Mediator and Observer patterns. Template classes are indicated by ⟨...⟩.

combination of other stoichiometric vectors s_j, with coefficients ω_j:

$$s_r = \sum_{j=0, j\neq r}^{N_{tar}-1} \omega_j \cdot s_j \qquad (19)$$

To check for linear dependent rows of S, we improve an approach by Luo et al.[18] In a first step, a reduced matrix $\tilde{S}$ is created by elimination of information concerning electrons and adsorbed species. Then, the rows of $\tilde{S}$ are subsequently filled into a test matrix A. If A^T contains at least a single colomn, the rank of A^T ($rank_A$) is determined by means of QR decomposition, every time an additional column is entered. A $rank_A$ value less than the number of columns in A^T indicates linear dependent stoichiometric vectors and, consequently, dependent reactions. For the calculation of the ω_j coefficients the last column of A^T is removed, obtaining $\tilde{A}^T$, and stored in a vector b. Then, $\tilde{A}^T$ has full rank, and the overdetermined linear system of equations $\tilde{A}^T \cdot \omega = b$ can be solved by means of the linear regression problem

$$\text{find } \omega \text{ such that } ||\tilde{A}^T \cdot \omega - b||_2 = 0 \qquad (20)$$

again employing the QR decomposition of matrix $\tilde{A}^T = Q \cdot R$,[43] with $||x||_2 = (x^T x)^{1/2}$. After some transformations of eq 20, ω can be computed by solving the linear system of equations

$$\tilde{R} \cdot \omega = Q^T b \qquad (21)$$

where $\tilde{R}$ is the upper triangular matrix within R. Direct solution is facilitated by the simple triangular form of $\tilde{R}$.

In general, all of these "complications" produce *many to many* relationships between thermodynamic and/or kinetic reaction parameter values. Moreover, if the described cases occur simultaneously within the same model, interdependencies between different sets of dependent parameters exist. Once a parameter value is changed, the values of all dependent parameters have to be updated.

An object-oriented design known to cope with such complex update relations is an interplay of the Mediator and the Observer patterns. The introduction of a mediating object reduces *many to many* to *one to many* relationships, which can then successfully be handled by the Observer pattern.[34] Figure 7a shows a typical object structure that occurs e.g. within EE_{disp} mechanism

$$A \rightleftharpoons B + e^- \qquad E^0_0, K_0 \text{ respectively (eq 18)} \qquad (22)$$

$$B \rightleftharpoons C + e^- \qquad E^0_1, K_1 \text{ respectively} \qquad (23)$$

$$2B \rightleftharpoons A + C \qquad K_2 \qquad (24)$$

where reactions 22−24 are dependent and thus K_0, K_1, and K_2 are related by general eq 17:

$$K_2{}^1 = K_0{}^{-1} \cdot K_1{}^1 = \exp\left[\frac{F}{RT}(E_0{}^0 - E_1{}^0)\right] \qquad (25)$$

Both reactions 22 and 23 have e.g. Butler Volmer rate laws with ButlerVolmerParameters objects. Disproportionation (24) is characterized by a power rate law with StandardRateParameters, representing kinetic constants and the equilibrium constant K_2 (for a discussion of the object-oriented design representing reactions, rate laws, and corresponding parameters see section 2.3.3). The TSRDirector acts as mediator and resolves the complex interdependencies between the parameter objects.

The according class hierarchy is shown in Figure 7b. Note, that the communication between parameters on the right-hand side and director and other classes on the left-hand side is controlled by two base classes, that set up the Observer pattern (in its standard form comprehensively described in ref 34). Furthermore, we enrich the update mechanism with certain ParameterAspects, which represent e.g. the name of the parameter and its changed value. A situation in which a client sets a new value of a kinetic constant is sketched by a pseudoline of code taken from the value() method of the StandardRateParameters class implementation. After the value is set, the inform() method of the subject base class is called and gets a KineticAspect as parameter, which constitutes a specific type of ParameterAspects. With a mechanism implemented in the Subject⟨⟩ class, only those observers who are interested in a change of kinetic param-

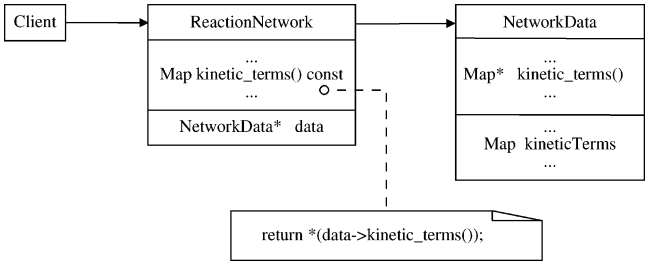

Figure 8. Proxy pattern in the implementation of the ReactionNetwork.

eters get an update signal. For example, the specific TSRDirector, which controls thermodynamic parameter values only, is not informed. Furthermore, detailed information about the parameter change is transferred by such aspects. For example, if a parameter that occurs in both a power rate law and a generic rate is changed within a StandardRateParameters object, information concerning the reaction index helps the GenericDirector to determine what generic parameter objects need to be supplied with new values.

2.3.3. Compilation Result. The compiler subsystem returns a ReactionNetwork object (Figure 4). The latter implements a protection Proxy in order to ensure secure access to the NetworkData from user code outside the kinetic compiler (Figure 8). A Client must not change the internal data generated during the translation process. Therefore, the ReactionNetwork class provides the same interface as the NetworkData class except that each function returns a copy of the according data structure and is provided with the additional read-only access qualifier const.

The interface of the ReactionNetwork class fulfills three basic requirements.

First, it provides functions that return the classified species and reactions in the network, mainly designated for usage of the ReactionNetwork object by a graphical user interface.

Second, for electrochemical experiments in which an excitation signal is applied to an electrode, it provides access to heterogeneous rate laws that can be coupled to the excitation function template library, described earlier.[1] Within the provided data structure, the laws are sorted by boundary segments, i.e., by boundary indices (see section 2.1) that describe electrode surfaces. Thus, for each electrode surface a unique excitation function can be applied for all relevant electrode reactions on that surface.

Finally, the ReactionNetwork gives access to the data structures that store the assembled kinetic and boundary terms, sorted by the index of species. This will be particularly important in the context of active simulation.

For the representation of reactions and species we choose a simple class design based on *object composition*.[34] Each Species object owns SpeciesParameters such as diffusion coefficients or initial concentrations. Reaction objects have rate Laws that again own ReactionParameters, such as rate constants or formal potentials.

Reaction and boundary terms are assembled for each species. This is accomplished with the class hierarchy shown in Figure 9 (Composite pattern).

In the trivial case, a reaction or boundary term is represented by a ZeroLaw, which returns the zero value for all time steps and all locations in space. However, if a species occurs within one or more rate expressions, a CompositeLaw is provided. Both classes inherit the abstract interface of the Law base class.

The add() function provided by all classes is part of the Composite pattern. Its implementation in the CompositeLaw enables a convenient summation procedure that composes all rate laws of the reactions in which the species occurs, possibly multiplied by the species' stoichiometric coefficient (int parameter in Figure 9):

$$\partial_t u_k = \sum_{i=0}^{N_r-1} \nu_{ik} \cdot r_i \qquad (26)$$

Equation 26 shows this procedure for homogeneous reactions, where N_r is the total number of homogeneous reactions in which species u_k occurs and r_i is the rate of reaction i. Note that we have declared the add() function as *pure virtual*. Therefore, all the derived Law classes must provide an implementation, although it might be pointless for the noncomposite classes (leaf classes). However, we follow the discussion in ref 34 and let the add function throw an exception if it is called on a leaf class object.

Among other methods, all classes give access to the function values and the gradients with respect to the species concentrations. This will be needed during numerical simulation.

For the *generic rate* concept described in section 2.1 a GenericRateLaw class is provided. The implementation of this class is based on the symbolic algebra library GiNaC.[46,47]

In particular, the creation and evaluation of the gradient, and thus of the Jacobian matrix rows, by an automatic differentiation algorithm is a crucial point within this context. Although symbolic differentation is less efficient, its use is indispensable for such a priori unknown kinetic expressions, to avoid truncation errors by finite difference approximations of the partial derivatives occurring within the gradient.

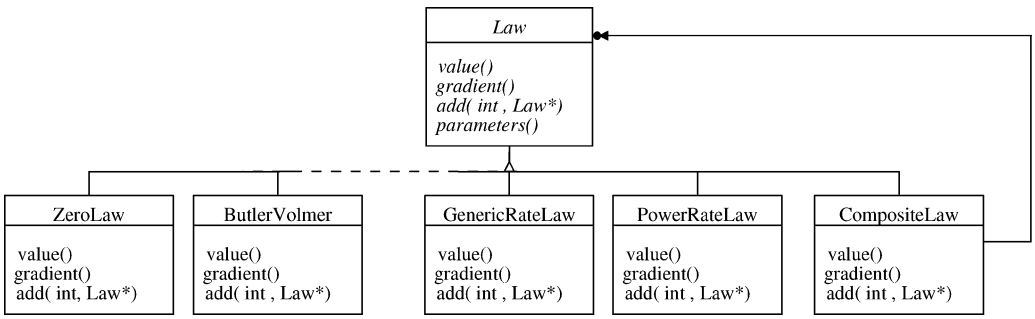

Figure 9. Composite pattern of the law concept. Only selected methods are shown. Pure virtual functions are written in italics.

```
   #include <ecco.hpp>
   using namespace ecco;
   ...
   // for a 1D model one has to provide a set of at least
   // two boundary indices, e.g. zero and one.
1  std::set<int> boundaries;
2  boundaries.insert(0);
3  boundaries.insert(1);
   // create the kinetic compiler object
4  KineticCompiler* kineticCompiler = new KineticCompiler();
   // start compilation and get a new ReactionNetwork proxy
5  ReactionNetwork* net = kineticCompiler->compile( filename, boundaries );
   // e.g. get the map of non-equilibrium reversible homogeneous reactions
6  std::map<unsigned int,Reaction* > dynHomReactions = net->dyn_hom_reactions();
   // or get the map of reaction terms
7  std::map<unsigned int,Law* > kineticTerms = net->kinetic_terms();
   // For other usage possibilities of the ReactionNetwork
   // object see section 2.3.3
   ...
8  delete net;
9  delete kineticCompiler;
```

Figure 10. Client code extract demonstrating the modeling pipeline provided by Ecco. The reaction mechanism can be provided by a text file named filename but also as input stream. The numbers on the left denote relevant code lines and are not part of the real source code.

However, the derivatives for rate expressions already known at compile time of the compiler itself, like those derived from the PowerRateLaw or from adsorption isotherms, are provided already within the corresponding class. Thus, minimal interference of the performance of the numerical solver coupled to the kinetic compiler is expected when running a simulation.

2.4. Application Strategies. Ecco's kinetic compiler is developed as part of the EChem++ project.[9] It belongs to the *Model* module, which includes all modeling and simulation tasks of the PSE.[1] Figure 10 shows a typical client code excerpt that makes use of the kinetic compiler of Ecco.

Lines 1−3 represent the requirements for the coupling of Ecco to a model geometry. It is assumed that such a geometry is framed by several boundaries that are identified by an index. A simple one-dimensonal electrochemical model encompases two boundaries: the electrode surface and the bulk solution (infinite distance for semi-infinite diffusion or a second electrode for a thin-layer cell model).

The two lines 4 and 5 cover the whole compilation procedure described in section 2.3.1. Note, that by calling the compile() function, a new instance of the ReactionNetwork object is returned. Therefore, the client has to delete that object after use by itself (line 8). This may be a small drawback since the new keyword is hidden inside the compiler facade. However, we choose this design since it enables a minimal interface of the ReactionNetwork that provides only those functions necessary to read out the network data.

In line 6, the client obtains a data structure that maps reaction indices to the appropriate Reaction pointer. With this pointer, it is able to access e.g. the according rate law and thus the reaction parameters (parameters() method in Figure 9). In an anologous way, the kinetic terms assembled for each species are requested (line 7).

Based on this procedure, the values of the model parameters of a specific reaction can be changed through a ReactionParameters object provided by the corresponding law.

Since the source code of Ecco is freely extendable, an additional application is discussed which shows how to modify the compiler. We sketch a situation that might arise if an additional adsorption rate law should be supported.

First, a new class has to be written which inherits the interface of the Law class (Figure 9) and implements the relevant functions according to the mathematical relation given by that law. A new keyword must be defined, e.g. "Temkin" for a law representing the Temkin adsorption isotherm.[48] This keyword must be added to the adsorptionLaw grammar rule (Figure 2). After that, the compiler accepts the string "Temkin" in the options section behind a reaction. Finally, the implementation of two synthesis objects has to be slightly customized: that of a preparator called AdsorptionLawPraparator to schedule the keyword in a data structure and that of a worker object called AdsorptionWorker (see Figure 4) to instantiate a new law object if the keyword was written within the mechanism behind an adsorption reaction. Recompilation will then integrate the changes into the framework.

2.5. Computational Aspects. The development of Ecco was performed under the Linux operating system (SuSE Linux 8.1, kernel version 2.4.19, and 9.1, kernel version 2.6.3, SuSE Linux AG, Nürnberg, Germany[49]) with the g++ compiler (versions 3.2 and 3.3) from the free GNU Compiler Collection.[50] The use of the widespread, mostly standardized and operating system independent C++ programming language[51] for the implementation will support Ecco's applicability by other working groups. The source code as well as the documentation is licensed under the terms of the GNU General Public License (GPL)[52] and can be obtained from the EChem++ project page.[9] The kinetic part of Ecco relies on the external open source libraries Spirit [37] and GiNaC[47] as well as on the STL.[42] We emphasize that the libraries greatly simplified the development of the kinetic part of Ecco, providing another example of code reuse made possible by clearly designed object-oriented software.

3. CONCLUSIONS

The kinetic facilities of Ecco extend and improve earlier approaches to (electro)chemical compilers. The component enables the modeling of a variety of (electro)chemical reaction mechanisms, including homogeneous and surface reactions and heterogeneous electron transfers as well as adsorption processes. General cell geometries are considered.

By default, the compiler assumes a power rate law for homogeneous and surface reactions. However, the *generic rate* concept offers the possibility to model and consequently simulate any kinetic relationship for homogeneous or surface reactions. Heterogeneous electron transfers and adsorption processes are assumed to follow common physicochemical models.

The input language reflects an intuitive chemical notation. The notational development of the language is not based on the mathematical model but on common formulations of chemical reactions optionally modified by specific kinetic expressions. The compilation of the reaction mechanism is fully separated from the subsequent simulation step. It is important to note that we have not made any specific assumption on what numerical method is used for the actual simulation. This opens Ecco to the implementation of any method of choice.

The use of well elaborated, object-oriented design patterns for the implementation of essential parts of the compiler both

provides an easy way to reuse the compiler within client code and reduces the effort to enrich the compiler with additional features, even by different programmers. In particular, the application of the command pattern within the compiler's synthesis part provides a convenient mechanism to add future algorithms. The facade pattern facilitates the use of the kinetic compiler from client code. It shields clients from the complex interplay of the objects inside the subsystem. This enhances clarity within software systems that reuse the kinetic facilities of Ecco and reduces computational costs during development. Furthermore, the modified observer pattern allows efficient communication between the various objects storing information about the reaction steps and species involved in the mechanism, ensuring consistent data within the system.

Based on the design and implementation decisions described in this paper, Ecco provides a flexible framework for the modeling of (electro)chemical processes. Additional formulations of process steps, rate laws etc. can be accommodated and coupled to the present implementation either without or with only minor changes to the basic code. For example, additional tasks which are mostly or completely independent of the reaction mechanism, such as the modeling of cell geometries, the choice of transport type within the system (e.g. diffusion, migration, hydrodynamics) can be added leaving the present code unchanged.

Beside its application to electrochemistry, reaction modeling with Ecco for the simulation of chemical reaction kinetics without heterogeneous electron-transfer processes is possible without restriction.

Making the source code of Ecco freely available under the terms of the GPL will overcome two major drawbacks of closed software systems in science in general and electrochemical modeling in particular: restriction with regard to an objective code validation as well as the need for reinventions.

ACKNOWLEDGMENT

We acknowledge e-mail correspondence with Hartmut Kaiser about the Spirit's parse tree behavior and with Richard B. Kreckel about GiNaC.

REFERENCES AND NOTES

(1) Ludwig, K.; Rajendran, L.; Speiser, B. *J. Electroanal. Chem.* **2004**, *568*, 203−214.
(2) Speiser, B. Numerical Simulation of Electroanalytical Experiments: Recent Advances in Methodology. In *Electroanal. Chem.* Bard, A. J.; Rubinstein, I., Ed.; Marcel Dekker: New York, 1996; pp 1−108.
(3) Georgiadou, M. *J. Electrochem. Soc.* **1997**, *144*, 2732−2739.
(4) Fulian, Q.; Fisher, A. C. *J. Phys. Chem. B* **1998**, *102*, 9647−9652.
(5) Fulian, Q.; Fisher, A. C.; Denuault, G. *J. Phys. Chem. B* **1999**, *103*, 4387−4392.
(6) Lee, H. J.; Beriet, C.; Ferrigno, R.; Girault, H. H. *J. Electroanal. Chem.* **2001**, *502*, 138−145.
(7) Sklyar, O.; Wittstock, G. *J. Phys. Chem. B* **2002**, *106*, 7499−7508.
(8) Bieniasz, L. K. Towards Computational Electrochemistry - A Kineticist's Perspective. In *Mod. Asp. Electrochem.* Vol. 35. Conway, B. E., White, R. E., Ed.; Kluwer Academic/Plenum Publishers: New York, 2002; pp 135−195.
(9) http://echempp.sourceforge.net, 2004.
(10) Bard, A. J.; Faulkner, L. R. *Electrochemical Methods. Fundamentals and Applications*, 2nd ed.; Wiley: New York, 2001.
(11) Svir, I. B.; Klymenko, O. V.; Platz, M. S. *Comput. Chem.* **2002**, *26*, 379−386.
(12) Kirkegaard, P.; Bjergbakke, E. *CHEMSIMUL: A Simulator for chemical kinetics*; Technical Report Riso-R-1085(EN); Riso National Laboratory, Roskilde, Denmark, 2000.
(13) Bieniasz, L. K. *Comput. Chem.* **1996**, *20*, 403−418.
(14) Rudolph, M. Digital Simulations with the Fast Implicit Finite Difference Algorithm - - The Development of a General Simulator for Electrochemical Processes. In *Physical Electrochemistry. Principles, Methods, and Applications*; Rubinstein, I., Ed.; Monographs in Electroanalytical Chemistry and Electrochemistry, Marcel Dekker: New York, 1995.
(15) Bieniasz, L. K. *Comput. Chem.* **1997**, *21*, 1−12.
(16) http://www.digielch.de, 2004.
(17) http://www.drhuang.com, 2004.
(18) Luo, W.; Feldberg, S. W.; Rudolph, M. *J. Electroanal. Chem.* **1994**, *368*, 109−113.
(19) Ketter, J. B.; Forry, S. P.; Wightman, R. M.; Feldberg, S. W. *Electrochem. Solid-State Lett.* **2004**, *7*, E18−E22.
(20) Bieniasz, L. K. *J. Electroanal. Chem.* **1996**, *406*, 33−43.
(21) Bieniasz, L. K. *J. Electroanal. Chem.* **1996**, *406*, 45−52.
(22) http://malina.ichf.edu.pl/zd-11/licence.htm, 2004.
(23) Missen, R. W.; Mims, C. A.; Saville, B. A. *Introduction to Chemical Reaction Engineering and Kinetics*; Wiley: New York, 1999.
(24) Wopschall, R. H.; Shain, I. *Anal. Chem.* **1967**, *39*, 1514−1527.
(25) Wopschall, R. H.; Shain, I. *Anal. Chem.* **1967**, *39*, 1535−1542.
(26) Feldberg, S. W. *Comput. Chem. Instrum.* **1972**, *2*, 185−215.
(27) Leverenz, A.; Speiser, B. *J. Electroanal. Chem.* **1991**, *318*, 69−89.
(28) Schulz, C.; Speiser, B. *J. Electroanal. Chem.* **1993**, *354*, 255−271.
(29) Lovric, M. *Anal. Bioanal. Chem.* **2002**, *373*, 781−786.
(30) http://www.opencascade.com, 2004.
(31) Ludwig, K.; Speiser, B. Manuscript in preparation.
(32) ISO/IEC 14977: 1996(E), Information technology − Syntactic metalanguage − Extended BNF, http://www.iso.ch,.
(33) de Guzman, J.; Kaiser, H.; Nuffer, D. C.; Uzdavinis, C.; Westfahl, J.; Arevalo-Baeza, J. C.; Wille, M. The Spirit Documentation v1.6.0, 2004.
(34) Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns. Elements of Reusable Object-Oriented Software;* Addison-Wesley: Boston, 1995.
(35) Ludwig, K. Manuscript in preparation.
(36) Aho, A. V.; Sethi, R.; Ullman, J. D. *Compilers. Principles, Techniques, and Tools;* Addison-Wesley: Reading, 1988.
(37) http://spirit.sourceforge.net, 2004.
(38) http://dinosaur.compilertools.net, 2004.
(39) http://www.gnu.org/software/flex, 2004.
(40) http://www.gnu.org/software/Bison/Bison.html, 2004.
(41) http://www.antlr.org, 2004.
(42) http://www.sgi.com/tech/stl, 2004.
(43) Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Fannery, B. *Numerical Recipes in C++, The Art of Scientific Computing*, 2nd ed.; Cambridge University Press: Cambridge, 2002.
(44) Opekar, F.; Beran, P. *J. Electroanal. Chem.* **1976**, *69*, 1−105.
(45) Deng, Z.-X.; Lin, X.-Q.; Tong, Z.-H. *Acta Chim. Sinica* **2002**, *60*, 1415−1418.
(46) Bauer, C.; Frink, A.; Kreckel, R. *J. Symbolic Comput.* **2002**, *33*, 1−12.
(47) http://www.ginac.de, 2004.
(48) Delahay, P.; Mohilner, D. M. *J. Am. Chem. Soc.* **1962**, *84*, 4247−4252.
(49) http://www.suse.de, 2004.
(50) http://www.gnu.org, 2004.
(51) Stroustrup, B. *The C++ Programming Language;* Addison-Wesley: Reading, 1997.
(52) http://opensource.org/licenses/gpl-license.php, 2004.

CI0497814