Representing Business Data Semantics In CIM using UML

Xiaofeng Wang, *Member, IEEE*, and Steve Van Ausdall

***Abstract*-- The IEC TC 57 Common Information Model (CIM) started with a focus on the power system models in the Energy Management Systems (EMS) domain. The need for a common model in Distribution Management Systems (DMS) and market operations drove CIM into those areas. The CIM was created in order to provide an information exchange format to facilitate the integration of applications and systems independent of their vendors and implementation. The current CIM model, including 61970 Part 301 and 61968 Part 11, intends to provide a standard semantic foundation for applications and systems to move toward interoperability. Given the fact that more and more integration projects are leveraging the standard CIM and using it as one of the input models forming the basis of an enterprise-wise model, the semantics (meaning "meanings") of the model are the key aspect that enables applications and systems (developers) to understand the data that is shared or exchanged among them. This paper provides an overview of how the business data semantics are represented in CIM using UML syntax. It elaborates on the meanings of the UML modeling concepts, which are further examined in the context of CIM. A set of recommendations is then proposed, regarding the modeling of CIM.**

***Index Terms*—Common Information Model (CIM), Unified Modeling Language (UML), Energy Management System (EMS), Distribution Management System (DMS)**

I. Introduction

THE common information model (CIM) is one of the most important parts of the IEC 91670 [1, Part 301] and IEC 61968 [2] series standards. The CIM provides a comprehensive logical view of a power system [1, Part 1]. CIM started with a focus on the power system models in the Energy Management Systems (EMS) domain [1, Part 301]. The need for a common model in Distribution Management Systems (DMS) drove extensions in the directions of asset management, distribution network, work management, and other related areas to be included in the CIM [2]. The new CIM Market Extension (CME) allows the CIM to be used as the basis for power market operations.

Before we proceed, it is important to understand what the CIM is, and its intended usage. First of all, the CIM is a structural (class / object) model that provides a standard way of representing power system objects in terms of classes and attributes, as well as the relationships between them [1, Part 1]. The comprehensive logical view represented by CIM is independent of any physical implementation languages and technologies. It focuses on modeling the electrical utility domain knowledge with logical modeling concepts so that a common model (semantic and syntax) can be achieved. Any extensions or modifications to the CIM model should not break this fundamental philosophy. Second, the usage of CIM is characterized as a tool to enable integration where a common model is needed to facilitate the interoperability regardless how each application or system is implemented [1, Part 1]. Because CIM is a logical model providing formal semantics for the electrical utility domain, it doesn't address the physical format and structures that an implementation model uses for data exchange and sharing. What it does do is specifying the domain concepts and semantics to which the implementation model can be mapped. Given this traceability back to a standard model, a mutual understanding of the data content being exchanged and shared can be achieved. The CIM is therefore a logical model where the business data semantics are represented.

Xiaofeng Wang and Steve Van Ausdall are with Xtensible Solutions, Satellite Beach, FL 32937 (e-mail: xwang@xtensible.net, svanausdall@xtensible.net)

II. Challenges

In order to promote and enable reusability, CIM defines power system domain knowledge using object oriented data modeling techniques specifying classes, attributes, and relationships in UML. The power system domain knowledge is then interpreted in the context of CIM so that when a term is referenced, a mutual understanding of the term can be achieved between different parties. Based on the business data semantics and requirements, the CIM modeling process typically involves identifying objects and relationships; designing classes, attributes, and associations; and abstracting class hierarchies. The most important goal in the CIM modeling process is to preserve and clarify the business data semantics so that the CIM can truly become the semantic foundation for systems and applications to share and exchange data.

This paper appreciates that a great deal of effort has been made to make CIM successful in facilitating interoperability. At the same time, this paper realizes that representing the business data semantics in CIM requires a clear understanding of the following:

- Meaning of the UML modeling concepts utilized – The meaning (semantics) of the UML modeling concepts provides a detailed descriptions of what a modeling concepts means including a set of properties (describing

difference aspects of the concepts), constraints on its uses, and its execution consequences [3, preface]. The meaning of the UML modeling concepts are the basic guideline of any object oriented data modeling practices.

- How the business data semantics are represented in the context of CIM – Modeling business data semantics and requirements with UML often results in a network of modeling concepts such as classes, attributes, and associations. The business data semantics are then carried by the modeling concepts and their semantics which provide a formal way to define entities, relationships, hierarchies, rules, constraints, and restrictions. With the formal representation of the business data semantics, the CIM then can serve as a semantic foundation not only for human understanding, but also for applications/systems to achieve common understanding, physical model (code) generations, validity checking, and other activities that formal semantics is required.

In the following section, the meaning (semantics) of the UML modeling concepts utilized in CIM are elaborated. At the same time, classes, attributes, associations, inheritance, and data types are explained and examined in the context of CIM in order to understand how business data semantics are represented. Recommendations are proposed for each CIM modeling concepts. This paper believes a set of normative rule regarding CIM modeling is much needed. Hopefully the recommendations proposed in this paper can provide a good starting point.

III. THE SEMANTICS OF CIM

Because UML has been chosen as the modeling language of CIM, a comprehensive understanding of the meanings of UML concepts (static view) is critical. This section elaborates the semantics of each modeling concept used in the CIM, including classes, attributes, associations, association class, inheritance, and data types. This section also provides an explanation and examination of how these modeling concepts are applied in the CIM model. This paper makes numerous references to the UML Reference Manual [3].

A. Class

Generally speaking, a class represents a concept in the domain being modeled. A concept could be a real-world object such as Pole, or an abstraction such as Asset. The classes representing abstractions are often used to describe common qualities shared by their descendents in inheritance hierarchies. Two main criteria determine whether a class is abstract. If the class represents a proxy for a concept outside the scope of CIM, (such as Person), it should be abstract. If the class is in scope, but it is not typically used without a context (such as Document), the class should be defined as abstract [3, Part 3 – Class].

Each class must have a non-null unique name within its container (such as package or containing class). A containing package or class defines a namespace. This defines the scope of a class name. The class name must be unique within its namespace. A class name may also contain a stereotype which is generally used for providing a usage distinction when code generation tools are applied [3, Part 3 – class name].

Based on the understanding the UML class basics, the semantics of classes modeled in CIM are examined and summarized as:

- Each CIM class shall provide an unambiguous description and definition of the concept it modeled. An exact domain meaning of each class is the foundation to guarantee that different systems and application understand CIM classes in a common way and use them in a consistent way.
- Because the CIM model doesn't explicitly define which classes are abstract [1, 2] (by default, all classes are concrete classes), specific implementations may differ in which classes can be instantiated and which can not. Furthermore, an ancestor class may be used to support substitutability principle (This is further explained in Generalization section). For example an application may want to use the PowerSystemResource class to represent any type of concrete power system resource in the situation that it doesn't know exactly what concrete class to use at the time. Either abstract or concrete needs to be explicitly defined for these classes.
- All CIM class definitions are grouped under certain packages. Although classes grouped in different packages may have the same name, each CIM class has a non-null unique name within the whole IEC 61970 Part 301 and IEC 61968 Part 11 model in order to avoid unnecessary confusions and ambiguities.
- All CIM classes (not representing data types) have NO stereotypes.

This paper proposes the following recommendations regarding CIM classes.

TABLE I
THE RECOMMENDATIONS FOR CIM CLASSES

UML Class Semantics	Recommendations
Abstract vs. Concrete	• Abstract vs. Concrete shall be addressed explicitly in CIM. • Each class shall be identified as either an abstract class or a concrete class. Recommendations are: o If a class is used for the substitutability principle which means it could be instantiated to represent concreted descendent classes, it should be set to concrete. Guidelines for how these classes shall be implemented shall be given within the scope of transforming CIM to an implementation model (This might be out of the scope of CIM modeling). o If a class is for the incremental description purpose only and will not be instantiated under any circumstances, it should be set to abstract.

Name and namespace	• A formal namespace for overall CIM including both 61970 and 61968 shall be given so that CIM concepts can be fully qualified and better referenced. • Each CIM class shall have a unique name within the CIM namespace. • From the syntax perspective, when the CIM model is transformed to other design artifacts (XSD, RDF, OWL, Java, C++, and etc), the class name can become a qualified name in most of the cases. A normative rule for naming convention shall be given. The following rules are recommended: o Exclude reserved words used in the popular implementation languages. o Use upper camel case (UCC). o Use character set of a-z and A-Z. o Don't use space, periods, or other separators.
Stereotype	• Stereotype shall not be used for a regular CIM class.

B. Attribute

An attribute is used to represent a particular aspect of an instance of a class. It is used to describe a value that an instance may hold [3, Part 3 – Attribute]. An attribute typically has the following settings:

- Type – Type specifies the type that the value of an attribute must conform to. Type can be a class name or a built-in data type (primitive). The actual value of the attribute must be an instance of the type of one of its descendants [3, Part 3 – Attribute, Type].
- Multiplicity – The multiplicity of an attribute specifies the how many values may be held by the object having the attribute. The usual multiplicity of an attribute could be exactly one, zero or one, zero or more, and one or more. If the upper limit is greater than one, it also indicates the sequence and uniqueness of the values [3, Part 3 – Attribute, Multiplicity]. The syntax of multiplicity is also defined in [3, Part 3 – Multiplicity].
- Name – Each attribute must have a non-null name. This name is used to identify the attribute in the class where the attribute is defined. The uniqueness of the attribute name (including the inherited attribute) needs to be achieved within the class [3, Part 3 – Attribute, Name].
- Scope – The scope defines whether the attribute is in the instance level or class level. By default, attributes are in the instance level. The semantics of the scope has been modified in UML2. The static feature flag on attribute is used to identify the scope of the attribute. For details, please refer to [3, Part 3 – Attribute, Static, Scope].
- Derivation – Derivation specifies if an attribute can be derived from other modeling elements. Although it may not add additional semantic meaning to the model, it may be used for clarification or designing purpose [3, Part 3 – Attribute, Derivation].
- Visibility – Visibility describes the capability of an attribute to be referenced by another modeling element in different container or class. Typically there are three status, Public, Protected, and Private. For details about the semantics for each status, please refer to [3, Part 3 – Visibility].
- Initial Value – Initial value specifies the initial value of an attribute when an object is instantiated. The initial value is expressed in a text string which will be used to interpret the value of the attribute when the object is initiated. For details, please refer to [3, Part 3, Initial Value].
- For the purpose of this paper, other settings like changeability, redefinition and subsetting, and tagged value are not elaborated here. For details about these settings, please refer to [3].

Based on the understanding of the UML attribute basics, the semantics of attributes modeled in CIM are examined and summarized as:

- Each CIM attribute (including both native and inherited) shall provide an unambiguous description and definition of the aspect it modeled. An exact domain meaning of each attribute is the foundation to guarantee that different systems and application understand CIM attributes in a common way and use them in a consistent way.
- Type – Each attribute defined in a CIM class (excluding these classes for data type purpose) has a type which is either a built-in type or a CIM data type class. So far, there is no attribute has been defined with a type as another CIM class.
- Multiplicity – The current multiplicity expression of all CIM attributes is omitted from the definition. According to the UML syntax [3, Part 3, Multiplicity], the multiplicity is exactly one when the expression is omitted. A discrepancy may exist between the model definition and the common understanding of the multiplicity of CIM attributes, which is zero to one if omitted.
- Name – Each CIM attribute (including both native and inherited attributes) has a unique non-null name within its containing class.
- Scope – The scope of all CIM attribute is at the instance level. There are no static attributes defined.
- Derivation – Derivation indicators are not used in any of the CIM attributes.
- Visibility – All CIM attributes are currently set to Public. Any status other than Public presents programming language independent information.
- Initial Value – Initial values have been seen in some of the CIM attributes.

This paper proposes the following recommendations regarding CIM attributes.

TABLE II
THE RECOMMENDATIONS FOR CIM ATTRIBUTES

UML Attribute Semantics	Recommendations
Type	• A normative rule is recommended to enforce that each attribute must have a type. • UML primitive and CIM user-defined data type are legitimate candidates for attribute type. • The possibility of modeling an association as an attribute (the type would be a regular CIM class) needs to be considered if the association is owned by the class.
Multiplicity	• Provides possibility of allowing absence of (optional) values or null values (zero multiplicity), and greater than one. • If the multiplicity is greater than one, sequence and uniqueness information needs to be given.

Name	• Each CIM attribute (including both native and inherited) shall have a unique name within its defining class. • From the syntax perspective, the attribute name should: o Exclude reserved words used in the popular implementation languages. o Use lower camel case (UCC). o Use character set of a-z and A-Z. o Don't use space, periods, or other separators.
Scope	• No static attributes are allowed in CIM.
Derivation	• Because derivation may provide some advantages for further clarification and saving computation, the possibility of using derivation shall be considered by the CIM standard.
Visibility	• All attribute must have "Public" visibility
Initial Value	• Initial value shall not be given to any CIM attributes. They should be used in the corresponding implementation model.
Stereotype	• Stereotype shall not be used for any CIM attribute.

C. Binary Association

In general, an association is a relationship among two or more specified classes that describes the connection between their instances. A binary association is an association that has exactly two association ends. The structure of a binary association typically includes:

- Association Name – An association may have an optional name which shall be unique among the associations and classes within the containing package. An association is not required to have a name. Rolename on its ends provides an alternative way to distinguish multiple associations among same classes [3, Part 3 – Association].
- Association End – Association end defines the participation of one class at a give position (role) in the association. Each association end specifies the properties that apply to the participation of the corresponding object. The properties include:
  - Name (rolename) – Association end name, also known as rolename, is used to identify the association end of an association. It is also used to navigate from one object to another object using the association. Given the purpose of the rolename, it must be unique in both namespaces (classes). All rolenames in an association must be different. The rolenames for a self-association are necessary to distinguish the roles that the same class plays on either end of the association. Rolenames are also necessary to distinguish associations between the same pair of classes which don't have association names. In the case that there is only one association between a pair of classes, the rolename is optional because the class name can be used to distinguish the association end [3, Part 3 – Association end, Rolename].
  - Visibility – The association end visibility specifies if the class on the far end can see the association toward the end with the visibility settings. The typical status of the visibility is Public, Private, and Protected. For more details, please refer to [3, Part 3 – Association end, visibility].
  - Multiplicity – Multiplicity defines possible number of objects for an association end that may exist simultaneously. If the lower bound is set to zero, a distinction should be made between absence of value and null. If the upper bound is set to greater than one, the sequence and uniqueness of the objects are also needs to be specified [3, Part 3 – Association end, Multiplicity].
  - Navigability – Navigability specifies if a given object on one end of the association is able to find the object(s) on the other end of the association [3, Part 3 – Association end, Navigability].
  - Aggregation – Aggregation models the whole-part relationship between an aggregate object and constituent objects. The object on the constituent side (part) is part of the object on the aggregate (whole) side. The additional semantics that the aggregation added to the association is that chains of aggregation instances may not form a circle. It is also important to understand that the lifecycle of constituent objects are independent of the aggregate object [3, Part 3 – Association end, Aggregation].
  - Composition – Composition is a stronger form of Aggregation with additional constraint which gives the ownership of the constituent objects to the aggregate object. Composition is used when the part objects are actually owned by the owner and don't have independent life without the owner. For details, please refer to [3, Part 3 – Association end, Aggregation, Composition].
- Derivation – Derivation specifies if an association can be derived from other modeling element.
- For the purpose of this paper, other settings like changeability, redefinition, and specialization are not elaborated here. For details, please refer to [3].

After understanding the basics of UML associations, the semantics of associations modeled in CIM are examined and summarized as:

- Each CIM association shall provide an unambiguous description and definition of the aspect it modeled. An exact domain meaning of each association is the foundation to guarantee that different systems and application understand CIM associations in a common way and use them in a consistent way.
- Association Name – There is no association name for any of the associations modeled in CIM so far.
- Association End
  - Name (rolename) – Each association end is given a name in CIM. From the semantic perspective, the namespace and uniqueness of each rolename need to be normatively defined.
  - Visibility – Each association end in CIM is set to public regarding the visibility.
  - Multiplicity – Each association end in CIM is defined with multiplicity information. Typical multiplicity includes exact one, zero to one, zero to many, and one to many.
  - Navigability – All the associations in CIM can be navigated in both ways.

- Aggregation and Composition – Aggregation is used to model the whole-part relationship in CIM. So far, there is no composition in CIM.
- Derivation – There is no association defined as a derived association. Actually derived association is the most commonly used derived element. It represents a virtual association that can be computed from two or more fundamental associations. For example a derived association can be set up between the ConductingEquipment and ConnectivityNode to show to which connectivity node a device is connect. The association can be computed from the relationship between ConductingEquipment->Terminal, and Terminal->ConnectivityNode. Although it may not provide additional semantic information, an implementation may want to explicitly include the derived association in order to avoid recomputing.

This paper proposes the following recommendations regarding CIM binary associations.

TABLE III
THE RECOMMENDATIONS FOR CIM BINARY ASSOCIATIONS

UML Binary Association Semantics		Recommendations
Association Name		• No binary association requires an association name.
Association End	Name (rolename)	• Each rolename (including both native and inherited) needs to be unique within its defining class.
	Visisbility	• Each association end shall have "Public" visibility
	Multiplicity	• If the lower bound is set to zero – The difference between absence of value and null shall be made. The recommendation is using absence of value. • If the upper bound is set to greater than one – Currently the ordering and uniqueness of a set of objects are not explicitly specified in CIM. In UML1, the ordering was separate from the multiplicity. In UML2, ordering and uniqueness are closely related to multiplicity. This paper recommends: o The sequence of objects shall be set to unordered. o The uniqueness of objects shall be set to unique.
	Navigability	• Possibility of defining one-way navigability needs to be considered in the CIM standard. Because navigability may implicate reference, pointers, or foreign keys in the implementation model, an appropriate one-way navigability may reduce unnecessary complexity for the implementation model.
	Aggregation and Composition	• Aggregation shall be used to model whole-part relationship. The chains of aggregation must not form a circle which mean if object (a) is part of object (b), object (b) can not be part of (a) directly or indirectly. • Composition shall be used in the case that the lifecycle of the part object is controlled by the whole object.
Derivation		• Because derivation may provide some advantages for further clarification and saving computation, the possibility of using derivation shall be considered by the CIM standard.

D. Association Class

A binary association provides the capabilities for modeling a relationship between two sets of objects. When the relationship needs its own properties, a binary association is not sufficient to provide necessary modeling capabilities. For example if the percentage of the ownership between a resource and an organization (a resource may be owned by more than one organization, an organization may own more than one resource) needs to be modeled, an attribute is probably needed to model the percentage for the ownership. This ownership percentage doesn't belong to either the resource or the organization. It belongs to the ownership relationship. An association class is usually applied to the scenario that the link between two or more objects need its own attributes, operations, or reference to other objects. An association class is an association that is also a class so that it has both association properties (association end) and class properties (attributes and operations). The association class and the corresponding association is really a single modeling element which describes all aspects of a link between objects.

From implementation perspective, an association class can be treated as a class with references to the association ends. This may lead to a way that decouples the association class (connecting class A and B) with a class that have binary associations to A and B. It is important to know that the semantics of the two modeling approaches are not the same although they may model the same information. If the uniqueness is set on the association end, the identity of an association class instance is determined by the unique combination of the references. If the setting on the association end is non-unique, the identity of an association class instance needs to be determined by properties of the association class along with the combination of the references. For more details, please refer to [3, Part 3 – Association Class].

Association classes are used in the CIM model [2]. Because association classes are associations and classes, all the settings for associations and classes needs to be given for any association class in CIM. In addition to that, the follow recommendations are proposed:

- If an association class is defined for an association between two classes, can the association be used without the involvement of the association class? Based on the semantics of the UML, the answer is no. The common understanding and usage on this case are yet to be established.
- If the multiplicity is greater than one, the uniqueness needs to be defined explicitly. Unlike the CIM binary associations, the introduction of the association class may also introduce the case that the pair of the instance of the two classes is not unique. For example the same pair of instances of Organization and Document may play different roles which are identified by difference instances of the DocOrgRole class.

- The identification of instances of an association class needs to be given. Because the association class is part of the association, the identification of instances of the association class is an important factor to fully describe the association.

E. Generalization

Generalization describes the relationship between a more general definition and a more specific definition. The more specific definition inherits from the general one and extends it. It fully complies with the general definition. There are two purposes of generalization:

- Substitutability Principle – This is to define the condition that an instance of a more specified class can be used when a variable of a more general class is declared. This rule enables polymorphism which is one of the most powerful object-orient programming concepts [3, Part 2 – Generalization].
- Inheritance – It is a mechanism of incrementally describe an element by sharing descriptions of its ancestors. With the inheritance, the subclass is able to incrementally define its properties by sharing the properties defined in its ancestors [3, Part 2 – Inheritance].

The substitutability principle provides the power of representing objects of descendent classes by the ancestor class. In CIM, some of the ancestor classes may never be used for substitutability. For example, the class Naming is purely designed for reusing the attributes and it is not appropriate to use it for representing any of the objects of the descendent classes. In most cases, the ancestor classes may be used to represent concrete descendent classes. Whether an ancestor class can be instantiated is depend on if it is abstract or concrete. Recommendations are given in the previous section regarding abstract vs. concrete.

For inheritance purpose, attributes and participations in associations are shared by subclasses. If a subclass is created, the inherited attributes and participations have the following semantics:

- Inherited attributes – All attributes defined in the ancestor classes may be shared by its descendents. All attributes including the inherited ones need to be unique in the descendent class.
- Inherited association participations – All association participations defined in the ancestor classes may be shared by its descendents. All association participations including the inherited ones need to be unique in the descendent class.

F. Data Type

In UML, there are primitive predefined data types and user-definable data types. The primitive predefined data type includes Numbers (Integer), Strings, and Booleans. The user-definable types are enumerations. The meanings of the primitive predefined data types are independent of any programming languages and they are not user-definable. The enumeration type is user-definable data type that has a name and a list of enumeration literals. In a particular implementation, a UML data type may need to be further expressed in a language type which is a data type expressed in a programming language (Java, C++, XSD, and etc). Language types are not defined in UML specification. A profile may need to be defined for language type for a particular language. For more details, please refer to [3, Part 3 – Data Type].

In CIM, data types are defined and used in the following ways:

- UML primitive predefined data types – String, Integer, and Booleans are used specify the type of attributes.
- Non UML primitive data types – Long, unsignedLong, Double, Short are also used to specify the type of attributes.
- Enumeration type – enumeration type is used to define enumerations.
- Stereotyped classes – stereotyped (<<Primitive>>) classes are used to define CIM data types. Usually, it has an attribute, value, to represent the value of the attribute, and another attribute, unit, to represent the engineering unit.

UML only defines a limited set of language independent primitive types. For the purpose of CIM, a richer set of data types including real number and date time which can provide a clear and programming language independent definition is much needed. Different profiles of mapping CIM data types to programming language specific data types may need to be developed. This paper proposes the following recommendations regarding the CIM data types:

- CIM data types shall not have identities which mean they are only used to represent values.
- Only UML primitive data type and CIM data type can be used as type for any attribute in the CIM data type classes.
- No associations shall be defined between any CIM data type classes.

IV. CONCLUSIONS

Because CIM is intended to be a logical model for facilitating the integration between applications and systems regardless of their implementation method, the semantics of CIM is a fundamental key factor to achieve that goal. To succeed, we need clearly defined domain knowledge, including concepts, relationships and rules to provide the foundation. Secondly, the domain knowledge needs to be represented consistently with UML modeling concepts and semantics in such a way that the domain concepts and relationships can be formally defined. This paper examines the existing semantics of the CIM model, and elaborates on the meanings of the UML modeling concepts used. At the same time, it explores how the business data semantics are represented within the context of CIM. The result is a set of recommendations proposing guidelines for the use of each CIM modeling construct to represent the business semantics.

V. Acknowledgments

The author of this paper would like to acknowledge the reviewers who made this paper possible. Joe Zhou, Shawn Hu, Terry Saxton, Greg Robinson, and Dan Martin provided valuable comments and feedback.

VI. References

[1] IEC 61970-301 Energy Management System Application Program Interfaces.

[2] IEC 61968 System Interface for Distribution Management – Part 11 Distribution Information Exchange Model.

[3] James Rumbaugh, Ivar Jacobson, Grady Booch, *The Unified Modeling Language Reference Manual Second Edition*, Addison-Wesley, 2005

VII. Biographies

Xiaofeng Wang received the B.S. and M.S. degrees in electrical engineering from Tsinghua University, Beijing, China, in 1995 and 1998, respectively, and the Ph.D. degree from the Electrical and Computer Engineering Department of Michigan Technological University, Houghton MI, in 2001. Currently, Dr. Wang is a solution manager in the Professional Services division of Xtensible Solutions and providing strategic integration consulting services to utilities worldwide. He has extensive knowledge and expertise in the integration technologies including SOA, Web Services, XML, RDF, OWL, and CIM. Dr. Wang started his professional career at GE Energy in 2001. Dr. Wang is a member of IEEE and IEC TC57 Working Group 14.

Steve Van Ausdall is a Solution Architect in the Professional Services division of Xtensible Solutions, providing guidance and development services to clients worldwide. He assists utilities using technology to optimize processes, extract insight from existing data and systems, and streamline data flows using standards-based enterprise-wide semantic model driven message and service based integration techniques. Mr. Van Ausdall has extensive knowledge and experience in enterprise data, application, and process integration technologies and products including service-oriented architecture, business process orchestration, geographic information management, energy and distribution management, asset management solutions, and performance management using business intelligence and dashboards. Steve has proven modeling experience, including UML diagrams, XML schemas, and business process models, and is proficient in the latest software development tools and methodologies. He draws upon this deep understanding of techniques to construct information infrastructures that will meet the functional, flexibility and reliability demands of the business.