KLEEN 1.1

White Paper

bdaum industrial communications

november 2006

Contents

Executive summary
4
Introducing Asset Oriented Modeling (AOM)
5
Open scenarios
5
The revolution of data models
5
Base concepts of AOM
6
AOM Benefits
8
KLEEN – a proof of concept
9
Extensible architecture
9
What KLEEN can do for you
10
Generating Code
12
Further reading
14
Platforms
14
Availability
14
Outlook
14
Contact
14

Executive summary

Within the life cycle of a project, implementation takes only 20 percent or less of the overall effort. Most of the effort and the budget go into the design phase. Most important during the design phase of an application is good communication between the partners evolved in the process, and that is what Conceptual Modeling is all about. All other things like code generation or reverse engineering are nice side effects, but side effects nevertheless.

While classical conceptual modeling methods such as Entity Relationship Modeling (ERM/ERD) or UML have proved to be adequate communication instruments for the established relational and object-oriented data models, this is not the case for the new, highly flexible, grammar driven data models as found in XML and other mark-up languages. As XML has taken the enterprise world by storm as an integration format for heterogeneous data, an adequate conceptual modeling method is badly needed.

Asset Oriented Modeling (AOM) has been developed with these requirements in mind. Easy to use, but based on a solid mathematical foundation it lends itself well to automated tool support. A first implementation of such a tool has now become available with the KLEEN modeler.

Introducing Asset Oriented Modeling (AOM)

Open scenarios

It seems that the closed world assumption of the enterprise data model has become a thing of the past. The business requirements of electronic business, supply chain integration, company mergers, outsourcing, and flat enterprise business models ask for open data models that can easily interoperate.

The revolution of data models

These new requirements have led to new data models, data models that are far more flexible, extensible, and dynamic than what we were used to in the past. The widespread adoption of XML as a *lingua franca* throughout the IT-community is a clear indication that the existing data models (like the relational and object-oriented data models) are far too rigid and narrow to cope with the new requirements. It is the *document* metaphor which characterizes these new data models. The governing construction principle is no longer the relational table or the hierarchical object but the *grammar*.

The revolution of data models has, of course, implications for conceptual modeling. Every time a new data model was introduced in the short history of computing, new conceptual modeling methods were introduced, too – methods that would cope better with the new data model. The introduction of the relational data model had sparked Entity Relationship Modeling, and the introduction of the object-oriented data model resulted in object-oriented modeling methods such as the UML and ORM.

While these modeling methods served very well for their respective data models, they are not equally appropriate for the new grammar based data models. The result is a semantic impedance mismatch between conceptual model and implementation, a mismatch that adversely affects the communication between the partners in the developing process. And, as you know, bad communication can be costly.

AOM removes this roadblock from dynamic application development.

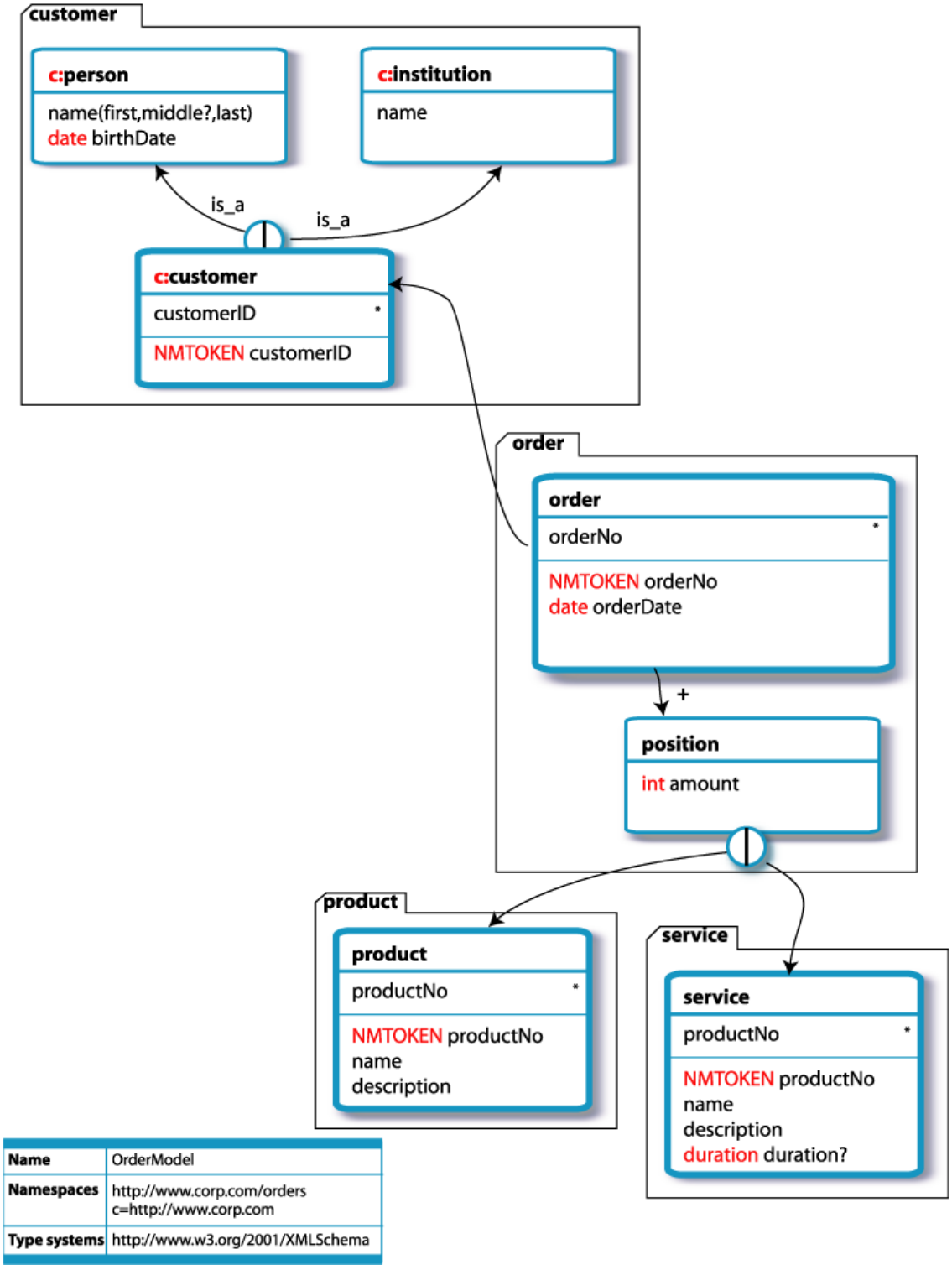
Base concepts of AOM

With Asset Oriented Modeling we propose a new modeling method that is more adequate to the new grammar based data models. AOM is based on four pillars:

- **A unified concept for entities and relationships.** Instead of representing entities and relationships with a different syntax, AOM represents both categories with a unified syntax element – the *Asset*. While this may sound unusual at first, it is actually very close to the relational data model, which also does not differentiate between entities and relationships: in a relational database both entities and relationships are implemented as tables. It was actually E. F. Codd, the father of relational algebra, who stated that there is no reason to distinguish between entity type and relationship type.

- **Higher order relationships.** The unified concept of assets makes it easy to formulate *higher order relationships*, i.e. relationships between relationships. Take for example the classical order model. When a customer orders a product, "orders" is typically modeled as a relationship between the entities "customer" and "product". A department may receive such an order. "receives" now is a relationship between entity "department" and relationship "orders". Another department may monitor the reception of an order. "monitors" now is a relationship between "department" and "receives". This is what we call higher order relationships. Classical Entity Relationship Diagrams cannot express such relationships between relationships. AOM, in contrast, builds on some of the concepts found in Bernhard Thalheim's *Higher Order Entity Relationship Modeling* (HERM), which provides a sound mathematical basis for higher order relationships.

- **Regular grammars.** AOM supports all data models that are based on regular grammars. For example, modern schema definition languages such as XML Schema or Relax NG are based on regular grammars. Thus, AOM is ideally – but not exclusively – suited as a modeling method for XML based implementations. At the same time it abstracts from the complex details of XML Schema.



AOM in action: this simple Order model shows some of AOM's key features such as multiple namespaces (the asset `c:customer` has been defined in a separate namespace). Note the definition of the complex property `name` in asset `c:person` and the definition of primary keys for the assets `c:customer` (customerID), `order` (orderNo), `product` (productNo), and `service` (productNo). Also, some properties such as `orderNo` and `productNo` are decorated with type declarations. AOM allows the use of various type systems: the XML Schema type system is used in this example.

- **Namespaces.** Each AOM model has its own – globally unique – home namespace but may contain assets from other namespace, too. This multi-namespace concept allows the merging of multiple models, an important feature in the open world of electronic business.

AOM Benefits

- **No difficult design decisions.** The decision if an item should be modeled as an entity or a relationship is always a difficult one. Take the previous example: should "orders" be modeled as a relationship "orders" or as an entity "Order" ? Taking the wrong decision may have negative consequence for the later extension of the model, and costly consequences when already large parts for the model are implemented. AOM does away with this uncertainty: in AOM everything is an asset. Actually, in the business world virtually any established business relationship is manifested in a business document, which, in turn, is an entity. The unified concept of the *Asset* reflects this reality. Less design decisions result in less wrong design decisions!
- **Extensible models.** The unified concept of assets allows easy extension of existing conceptual models. This is beneficial in dynamic environments where data models change frequently.
- **No semantic impedance mismatch.** The support for grammar based data structures allows exploiting the full power of modern mark-up languages such as XML without loosing support for legacy data models such as the relational and the object-oriented data model.
- **Compact models.** AOM's notation for complex data structures allows very compact models, indeed. This not only allows creating models quickly, but also produces models that are more comprehensive and can easily be reviewed. Communication is improved.
- **Model integration.** AOM's support for namespaces and its ability to merge bind multiple models into one allows the modular development of models and the easy integration of formerly independent models. The ability to define bindings between models encourages the use of conceptual Design Patterns.

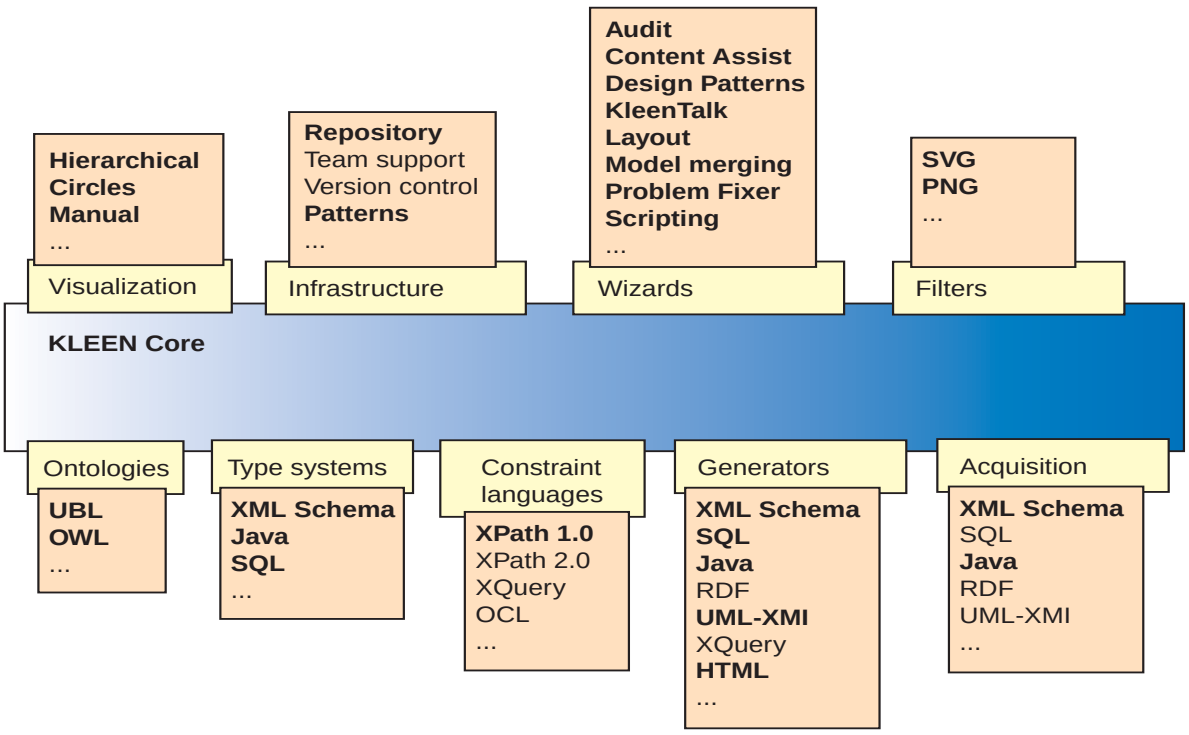
KLEEN – a proof of concept

Asset Oriented Modeling is designed as a general modeling method. The most basic tools to use it are pencil and paper. However, the solid mathematical foundation of AOM makes this method an ideal candidate for intelligent tool support. This is why we implemented KLEEN.

The name KLEEN was chosen to honor the work of mathematician Stephen Cole Kleene (1909-1994). Kleene worked dominantly on the theory of algorithms and recursive functions. His work on recursion theory helped to provide the foundations of theoretical computer science. Among many other things Kleene developed the theory of regular sets which now forms the mathematical basis not only for regular expressions in programming languages, but also for the task of schema definition in modern markup languages such as XML. Asset Oriented Modeling (AOM) and KLEEN are strongly based on those theories.

Extensible architecture

In the current implementation state KLEEN delivers only a glimpse of what is possible with AOM. But KLEEN features an extensible architecture, allowing plug-in support for various type systems, constraint languages, design wizards, filters, and application generators.

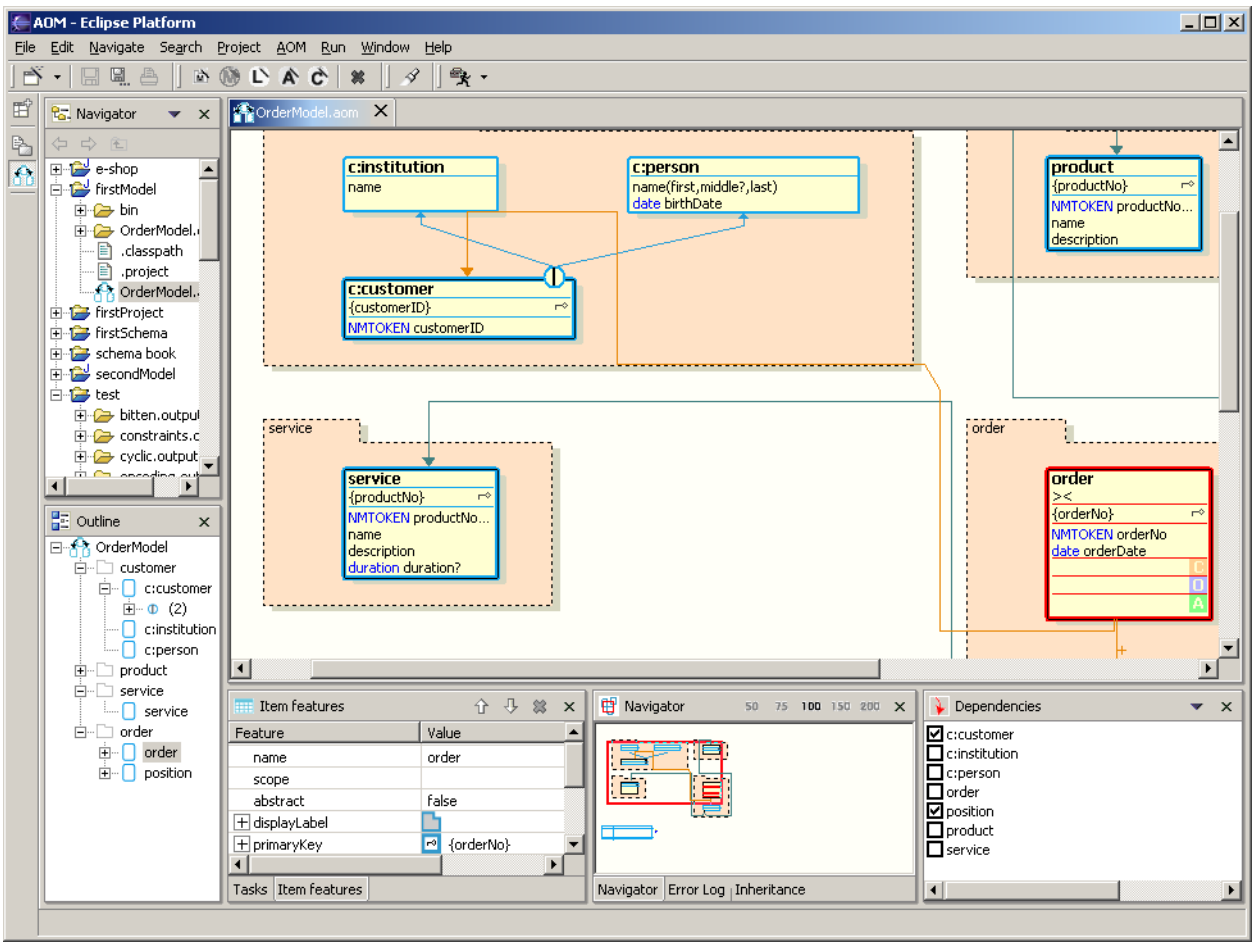


KLEEN's plug-in architecture is highly extensible. Currently implemented are the plug-ins printed in bold.

What KLEEN can do for you

In the current implementation state KLEEN provides the following features:

- **Diagram drawing.** KLEEN features a pluggable gaph visualization concept. Automatic layouts such as the Hierarchical Layout and the Nested Circle Layout take care of all layout decisions allowing designers concentrating on the conceptual aspects of modeling while the Manual Layout provides full control about the design, Diagrams can be exported as SVG and PNG graphics. Tiled printing is supported, too..
- **Asset illustration.** KLEEN allows attaching images to assets as a visual cue for easy asset identification.
- **Ontology support.** KLEEN allows you to connect your models to ontologies thus ensuring that your models conform to internationally standardized business terminology. Currently KLEEN supports the UBL (Universal Business Language) from OASIS und the OWL (Ontology Web Language) from the W3C.
- **KleenTalk**. If you prefer a text description of your model, here it is. KleenTalk allows specifying a model in plain English sentences. KLEEN automatically converts such a description into a diagram.
- **Syntax check for property expressions.** KLEEN supports the full AOM syntax for property expressions and instantly checks property expressions for valid syntax and type declarations.
- **Type systems.** Currently, KLEEN supports three extensible type systems: XML Schema, Java, and SQL. A model may opt to use a single type system, multiple type systems, or no type system at all.
- **Constraint languages.** Currently, KLEEN supports XPath 1.0 as a constraint language. Constrained expressions are checked for valid syntax.
- **Instant sanity check.** KLEEN instantly checks the model for sanity after each operation. Cyclic structures, orphaned assets, and other design problems are detected and recorded in the task list (see below). KLEENs Quick Fix feature proposes solutions for most of problems.
- **Content wizards.** KLEEN's content wizards help the user to formulate property expressions, constraints and other expressions..
- **Semantic operations.** Beyond the basic operations such as the creation and removal of assets and arcs, KLEEN also implements complex operations that exploit the specific semantics of AOM. Among those are: Resolve Inheritance, Explode Asset, Implode Asset, and Arc Reversal. These operations allow designers to modify models easily in a highly automated fashion.



KLEEN is currently implemented as a plug-in to the Eclipse platform.

- **Audit wizard.** The audit wizard can guide through the definition of a new model, or can perform an audit for an existing model. This wizard detects redundancies and helps arriving at a complete, normalized model.
- **Model merging.** KLEEN allows to combine multiple independent models into a single composite model. This feature allows for a modular development of models. Binding definitions can be used to mold fairly heterogenous submodels into one consistent composite model.
- **Design Patterns.** KLEEN supports the definition and application of conceptual Design Patterns and comes with a library of built-in patterns.
- **Task List.** The task lists contains unfinished tasks and problems. It thus provides an up-to-date overview of the current state of the model and what remains to be done.
- **Navigator.** A navigator view provides a bird's eye view over the complete model and allow easy scrolling within the model.
- **Search function.** A powerful search functions allows to search for constructs within a model and across multiple models within the whole workspace..
- **Acquisition.** Foreign definitions such as XML Schema type libraries can be imported into the model.

Generating Code

KLEEN is able to generate code straight from the conceptual model. As a matter of fact, the code quality depends on the level of detail provided with each model. The KLEEN annotation mechanism allows adding such detail in form of implementation notes and thus to influence the generation process.

Currently, the following generator plug-ins are available:

- **XML Schema generator.** The XML Schema generator can derive XML Schema code from a completed model. Since some higher order AOM concepts such as multiple inheritance, non-deterministic expressions, and multiple namespaces are not supported by XML Schema, the schema generator translates those AOM structures into equivalent XML Schema structures similarly as a higher programming language is translated into assembler. KLEEN can thus save many hours of schema authoring. A simple scripting language allows configuring the schema generator for specific purposes. Types from foreign type systems (Java, SQL) are automatically converted to equivalent XML Schema types. Multiple inheritance relationships are resolved. Non-deterministic expressions are transformed into deterministic expressions as required by XML Schema, and each namespace used in the model results in an own schema file with a matching target namespace.

- **UML-XMI generator.** The UML-XMI generator can translate the AOM model into an XMI 1.0 file which can be imported by various UML tools. This allows to utilize KLEEN in environments that use UML as a modeling method for implementation models.

- **SQL generator.** The SQL generator translates AOM models into SQL database schemata. The generated set of CREATE TABLE and CREATE INDEX statements implements the model as a relational database. Complex properties are either embedded or implemented as subtables depending on preferences. Arcs and cardinality constraints are transformed into integrity constraints (FOREIGN KEY and CHECK statements). Various RDBMS dialects can be selected to support a wide array of real world database systems.

- **Java generator.** The Java generator allows to generate code directly from the conceptual model. The tedious process of implementation modeling can be completely sidestepped. KLEEN's Java generator can produce ready-to-run applications that only require the business logic to be filled in.
  The Java generator can generate a complete Java class hierarchy from the AOM model definition. Access methods for class fields are equipped with integrity checks. Optional flat or bubbled event mechanisms can inform listeners about changes in the data model. Even constraints formulated in XPath 1.0 are translated into Java source code.
  In addition to these data structure, KLEEN's Java generator can generate a framework of loosely coupled program modules (aspects). All what remains to do, is to fill in the blanks with business logic, to instrument the application with these aspects, and to run the application. Later on, business logic modules can be easily exchanged or added.
  Using these concepts, KLEEN avoids the hassles of round-trip engineering and allows for quick modification of both model and application.
  If persistent data structures are required, KLEEN's Java Generator can produce marshalling and unmarshalling routines which can transform Java data structures into XML and vice versa. As a matter of fact, these XML documents are compliant with the schemata generated with the XML Schema generator. Optionally, the Java generator can produce Hibernate metafiles for storing data models in a relational database.
- **HTML generator.** This generator allows to produce an HTML documentation of the model. Optionally an SVG drawing of the model can be included. XSLT stylesheets allow customizing the generated pages.

Further reading

An introduction into AOM is found on the AOM website at

www.aomodeling.org and in the recent books

System Architecture with XML; Berthold Daum and Udo Merten;

ISBN: 1-55560-745-5; Morgan Kaufmann Publishing

and

Modeling Business Objects with XML Schema; Berthold Daum;

ISBN: 1-55860-816-8; Morgan Kaufmann Publishing

Platforms

KLEEN is currently implemented in Java as a plug-in to the Eclipse 3.0 platform. This allows KLEEN to run under Windows, Linux, Mac OS X, Solaris, QNX, AIX, HP-UX, and other operating systems.

Availability

KLEEN is available now. The current status of the product is "production" on Windows, "beta" on all other platforms. The community version of KLEEN is available for free, extended facilities are provided on a commercial basis.

Outlook

KLEEN is an ongoing project. Among the tasks on the planning lists are – apart from improvements to the user interface – the support of further constraint languages, the implementation of additional generators, and the provision of business pattern libraries. The priorities depend more or less on you. Tell us what is most important to you in a contemporary conceptual modeling tool. We are listening.

Contact

bdaum industrial communications

info@bdaum.de

info@aomodeling.org