Presentation for InterexWorks 2000

Title: The Role of XML in Automated System Design

Author name: J.A.J. (Hans) van Leunen
Company: Philips Semiconductors
Adress: Building BE 519; PO Box 218, 5600 MD, Eindhoven, The Netherlands
Phone: 31 40 2722372/2722198
Fax: 31 40 2722764
Email: Hans.van.Leunen@philips.com

Impact of XML Technology

The Role of XML in Automated System Design

Introduction

The eXtensible Markup Language (XML) is a plain text format in which hierarchically structured information can be stored. The language is a derivative of SGML and has close relations with HTML. Together with associated language standards it provides for an optimal information exchange format for information that is residing on web pages. However its use is not limited to the Internet. Many tools that are not directly web related make use of this easy interchange service. Amongst them are electronic program guides and MPEG-7. XML will have its greatest impact on web based and directly web related technology. For example data warehousing technology will extensively use XML. In this paper the attention will be guided to the impact of XML on automated system design technology. The domain treated will encompass both hardware and software technology. Due to the field of expertise of the author, a senior system software architect, the weight will be slightly more at the software side.

The paper will start with an introduction into modular system design technology. This is necessary in order to understand the role that XML can play in supporting automated system design[1]. Next the capabilities of XML are treated. This will be focussed on the capabilities of XML with respect to distributed data exchange and to the role that XML is playing in the interchange of design information between tools and between tools and publicly accessible repositories. Finally some potential scenarios are given in which possible applications of XML based technology are sketched.

Content

[1] A more in-depth treatment of modular system design technology can be found on http://sites.netscape.net/hansvanleunen

Driving forces in system generation technology

There are two main driving forces that influence the trend to switch from monolithic to component-based systems. These are the dichotomy[2] of architectures and relational clarity.

Relational clarity

The first driving force is relational clarity. Relational clarity is best explained via its antonym, potential relational complexity. The number of relations that an uninitiated novice is confronted with, when he meets a new system for the first time, characterizes the potential relational complexity. It is also the complexity that a symbolic browser or a reverse engineering tool would have to cope with. This indicates that relation clarity has a direct relation with the manageability of a system. Relational clarity becomes important when a system must be reconfigured, extended or maintained. The relational clarity of a four-layered architecture is typically 30% better than the relational clarity of a single layer system. This is caused by the fact that calls to deeper layers than the next layer are not directly supported. It is also caused by the fact that backward communication calls are normally prohibited. They are served by a special callback mechanism that usually works on a subscription basis. In contrast to this, the relational clarity of component-based systems is typically some orders of magnitude better than a corresponding single layer monolithic system. The consequence is that large component based systems are much better manageable and maintainable than corresponding monolithic systems.

In both hardware and software the success of a choice between architectures is highly dependent on the clarity at which interfaces of layers or components are described. Reusable components are always accompanied by a clear, complete and reliable description of their interfaces.

In software, the relational clarity of object oriented solutions may range from worse than flat API's to as good as component based systems. The key factors here are how much use is made of deep inheritance (-) and how well encapsulation is implemented (+). C++ does not require strict encapsulation. Software component technology makes a point of correct and hard encapsulation. The effectiveness of encapsulation plays an important role not only for the manageability of the target system but also for the reusability of the software components.

Dichotomy

Another important driving force is a feature that is shared by all architectures. Architectures can be divided into two parts; a passive relational part and an active part. This occurs not only with architectures. It is a feature of all subjects that have to do with modeling. For example, when a new organization is planned, then first a sketch of all departments and their interlinks is made. After that, their functionality is described. At the end, the organization is realized and started by installing the people that will operate that organization. The dichotomy does not split the architecture into two complementary parts but the division is clear enough to indicate what elements belong in what part. The passive relational part can be completely described by information that is publishable. At the same time it contains sufficient information to construct useful sketches or even testable skeletons of system components. In general, specification of the active part is orders of magnitude more complex than specifying the passive relational part. The active part contains information that one may want to hide. It represents the intellectual property of the system designer. Both hardware component technology and software component technology provide proper means to encapsulate and hide this IP. The information contained in the passive relational part is needed to promote the product and to describe the working of the product in user manuals and reference manuals. If the information is made available in machine retrievable format on publicly accessible repositories, then this

[2] Dichotomy is a characteristic of an item, which states that the item can be split into two discernable parts.

service may be used to create testable skeletons of the published components. The applicability of the component in the target system will become clear very fast, especially when this decision is supported be the toolkit. When the decision turns out positively, then an attached URL in the same information set may be used to order the component. Together these measures will install a busy open market for components that can be used in the configuration of modular systems.

System modeling

Modeling elements

Where painters use colors and forms to generate an abstraction of their subject, system architects will use properties, aspects of behavior, relations, communication, encapsulation and coordination as ingredients for their model

Modularization and encapsulation

One of the modeling elements is encapsulation. In order to enable the application of this modeling element the main model must first be subdivided in a series of sub-models. On their turn, the sub-models can also be divided into sub-models. This process can be continued until non-dividable sub-models are reached. The models and sub-models that contain encapsulation as one of their modeling elements will be considered as objects, while the other models will be considered as patterns. In the following discussion, only objects are treated. The described process enables a subdivision of the labor necessary to create a large and complex system. However, this subdivision has only sense when the system is divided into modules that are sufficiently independent from each other. In that case, the creation of the corresponding modules can be delegated to independent contributors. The fact that a sub-model is independent from other sub-models means that it can be effectively encapsulated. In that way, the relations with other sub-models are minimized. Usually a small set of relations to other sub-models results. These relations are used to communicate commands and or data. A sub-model that fulfils these requirements is a component of the grand model.

Independent modeling elements

In creating automatic systems, the following natural modeling elements can be discerned:

1. Properties
2. Relations
3. Aspects of behavior
4. Communication
5. Encapsulation
6. Coordination

The first four of these elements are interdependent. It is possible to transform this subset into a new set that is no longer interdependent. Each modeling element in this set is characterized by two requirements:

1. The modeling element is conceptually clear.
2. The modeling element has a direct or indirect implementation via the methods and tools of the system or component builder.

This last requirement is in line with the statement that an architectural view has little value when there is no direct or indirect way to implement that view. Properties and relations can be categorized as attributes that together constitute the internal state of the item that is being modeled. Behavior is often characteristic for a group of items that behave in a similar way. The internal state of the individual governs its behavior. Thus, behavior divides into two modeling elements: the internal state and class wide behavior. By adopting singleton classes, this rule transforms into a generally valid statement. The term operation is used for an aspect of class wide behavior.

The communication path is a set of relations. The transferred message or the transferred command is a set of attributes. Communication must take place via a predefined protocol and after arrival, the message or the command causes a trigger. This trigger activates an operation. The attributes contained in the message or command form the parameters in the call of the corresponding operation. Like properties and relations, these parameters can also be categorized as attributes. The trigger may cause a reaction or the operation may just change the internal state of the callee. Therefore, a new set of mutually independent modeling elements can be derived from the original set of natural modeling elements:

1. Attributes
2. Operations
3. Protocols
4. Triggers
5. Encapsulation
6. Coordination

Operations are class wide concepts. Protocols should have an even wider scope. Triggers and attributes can be both individual and class wide. Encapsulation works for the individual, but the individual also encapsulates its class. Coordination still is a complex issue. Coordination encompasses threads of execution, synchronization and events. Coordination can be centrally instigated, but usually coordination takes place between individuals. If use is made of these characteristics, then the architecture will reflect these choices.

In software architectures, modeling elements 1 through 4 represent a purely functional approach. If encapsulation is included as well, then an object oriented or a component oriented approach results. Some of the most modern programming languages or tools tackle also the sixth modeling element. Older languages and tools rely on services delivered by the operating system or by a real-time operating system kernel.

Implementation of modeling elements in hardware

The discussed independent modeling elements have feasible implementations in hardware. However, in this domain the possible implementations are rather multiform. Still attributes constitute the internal state. Operations represent a class wide aspect of behavior. Protocols specify how communication is done. Triggers start activity and coordination controls parallel or sequential processing of actions. Encapsulation has a rather visual meaning for hardware devices.

Implementation of modeling elements in software

As shown, the original, more natural modeling ingredients convert in a new set of mutually independent categories of modeling ingredients. Programmers have straightforward implementations for each of these new ingredients.

Modeling element	Category	Implementation example
Attributes/state	Passive relational	Fields of data structures

Operations	Active	Active code of routines
Protocols	Passive relational	Language documents, interface specs, skeletons of routines
Triggers	Active	Event of calling the routine
Encapsulation	Passive relational	Abstract data type
Coordination	Active	RTKOS service, threads, events, synchronization primitives

Attributes are implemented in fields of data structures, which in their turn are reserved areas in the available memory space.

The implementation of operations is independent of the state of the individual. A set of attributes represents this state. The routines implementing the operations use a reference to this set of attributes as an input parameter. In this way, they themselves become independent of that state.

Protocols are defined by the language, which is used to program the routines and are further specified by the function prototypes of these routines.

A trigger represents the event of calling a routine. The parameters of the call represent the attributes that are contained in the message or command. The trigger may cause an immediate response or the effect may be limited to a change of the internal state of the callee.

Encapsulation can be achieved by applying abstract data types (ADT's). Abstract data types consist of a data structure that contains all attributes that characterize the individual. It also contains one or more references to data structures that contain class wide attributes and / or references to implementations of operations. Most programming languages support dynamic extension of data structures. This can be exploited in implementing a derived class by extending the data structures of the parent class, or by exchanging references to existing operations with references to more sophisticated implementations of that same operation. The new version of the implementation of the operation may make use of the extra attributes. Object oriented languages provide these services invisibly and without further interaction of the programmer[3].

Third generation languages do not directly support coordination. Instead, services from a real-time kernel operating system (RTKOS) implement coordination. Some modern programming languages offer direct support for some aspects of coordination. For example, Java supports threads and the keyword: synchronize. Standard Java packages support event mechanisms.

Organizing reuse

Ordering

If all models and sub-models that occur in an application domain are ordered with respect to their complexity, then it becomes possible to find models or sub-models that can be considered as variations or extensions of other models or sub-models. If this is exploited, then the implementation of existing models can be partly or completely reused. Usually the most simple models or sub-models are most frequently used. So it is sensible to start with the simple models and then try to derive as much as is possible or manageable the more complex models from the already implemented work. This works best if the implementation of the simpler models is prepared for this purpose.

[3] The programmer gives guidance by specifying the class structure in a header file.

Equivalence classes

If all sub-models are analyzed, then they can be grouped, such that in each group the group members expose similar behavior. In this way, the groups form equivalence classes. Operations represent aspects of class wide behavior. The actual behavior of a sub-model is determined by the operations and by the set of attributes that describe the current state of the sub-model. Apart from that, the behavior is influenced by information consisting of parameters that accompany the trigger that activates the operation. The attributes that constitute the internal state can be interpreted as hidden parameters. Parameters can be input parameters or both input and output parameters. In hardware, an input parameter describes the signal that enters an input port or a sensor. The output parameters describe the output signals. In software the input and output parameters are present in the call of the routine that implements the operation. The return value is an input-output parameter. Its input value is part of the internal state and contains a default or initial value. The advantage of the classification process is, that class-wide aspects have to be implemented only once per class. When complex classes can inherit functionality from simpler classes, an even higher degree of reuse can be obtained. This is one of the best values of the object-oriented approach.

Meta-modeling elements

Classes are described by a type definition. These type definitions are often referred to. It means that the type definition and the type reference are playing a role that gives them the status of a meta-modeling element[4]. Operations are often grouped in interfaces. If this is done in an orderly and organized fashion then also the interfaces are going to play the role of a meta-modeling element. In fact these examples present items that will be reused much more frequently than the modeling elements from which they are derived! This is a general characteristic of meta-modeling elements. If a designer uses interfaces and types that are already in use by a large community, then the chance that he will end up with an optimally reusable product is much larger than when he decides to invent his own interfaces and does not bother with available types.

Architecture

An architecture is a set of views of a model

System creation is an art of modeling a natural or artificial target.

Creating a product runs through several phases.

1. Motivation. Evaluation of the reasons why the creation process must be started.
2. System conception. Analysis of the basic requirements of the target.
3. System design. Detailed analysis and successive implementation of the requirements.
4. Building or acquiring the parts of the system.
5. System integration. Final integration and final accomplishment test.
6. System support and maintenance.

The model changes dynamically during the traverse through each of these phases.

An architecture is a set of structured views of a model. The views represent different abstractions of the current model and are based on established rules and methods and on

[4] Meta-models describe lower-level modeling elements. Thus meta-meta-modeling elements describe the elements of a meta-model.

the customer's requirements. The current model itself is an abstraction or a partial realization of the required target product. A single architect or a group of cooperating architects is responsible for the generation of these abstractions. The set of requirements that are available to the architects hardly ever cover the complete customer requirements. In large and complex projects, the set of underlying rules is often incomplete or even inconsistent. This means that the creative and combinatorial talents of the involved architects influence the views. The system builders use these views as guides in the system creation process. For this reason, the architects have a significant influence on the resulting architecture.

In each of the phases of the system creation process, several structured views may visualize the current and / or planned state. These views together constitute the architecture of the current phase. In the beginning the views are rather sketchy. Later the views become much more detailed.

An architectural view may consist of a set of pictures. An architectural view may also exist of a text-based document. Text-based documents can be much more detailed than a pictorial view. In a text-based document, lists and tables support the structure of the view. Often browser tools help navigating the information contained in a set of architectural views. These tools may be able to generate derived views that give a different or more abstract insight.

The views highlight the division of the target into relational and / or functional components. Multiple usages of similar or nearly similar patterns, components or subsystems will become apparent. In each of the views, reusable design aspects are highlighted in a recognizable way. For this reason, the architectural views are an excellent means to promote reuse of existing work. Reusability can be realized within the same project or between equivalent projects.

The value of an architecture

An architectural view has no value when there exist no direct or indirect means to convert the view in an implementation of the described concepts. With other words, the views must transform manually or automatically in a recipe for building or using a part or the complete item or equivalent copies of that item.

If the maintenance and support of architectural documents must be minimized, then each view must be made as independent from all the other views as is reasonably possible. In that way, a change in one of the documents does not necessitate a corresponding change in other documents.

A complete set consisting of mutually independent views delivers the most efficient use of an architecture. This is the case when the set covers all aspects while at the same time the contents of one of the views can be changed without affecting any of the other views in the set.

The notion of mutual independence is not straightforward. A workable choice is to consider two documents independent if their subject differs, while at the same time the subject is conceptually clear, and finds a direct implementation via the methods and tools that the builders apply. A mutually independent set of views can be derived from a corresponding mutually independent set of modeling elements.

Dynamics of the architecture

When a project runs through its phases the corresponding architecture will change dynamically. During the process of designing and building an item, several different versions of the architecture play a role. The new version may contain a different set of views. The views that stay usually contain more detail. The result is that early versions of an architecture represent a rather vague and sketchy description of the target item, while the final version is a rather elaborate and close description of the resulting product. One of the final views is the user manual or a help file. Another final view may be the reference manual. Thus, with this respect, final documentation is also a part of the architecture.

About XML

Capabilities of XML

XML, the Extendable Mark-up Language, is quickly becoming the next hype, so it is good to have a proper idea where this is all about. What is the real significance of XML? How does it differ from software component technology or object-orientation? It is possible to give some fundamental answers to these questions!

An XML file is a container that can be filled with a hierarchically structured set of data. Tags mark the hierarchy. Similar to HTML, the tags and the data are written in plain text. In contrast to HTML the tags can be chosen freely. De meaning of the tags can, but must not be defined in a special Data Type Definition (DTD) or schema[5] file that is associated to the XML container file. The revolutionary aspect of XML is, that for the first time such a hierarchically structured container is specified by an internationally accepted standard. The ISO XML standard is prepared by the workgroup W3C and is accepted in February 1998. Another important aspect is, that the content of XML files is easily accessible both by tools and by end users. In order to support the easy readability for the end user another associated file type, the eXtensible Style Language file is available in which can be specified how the contained data must be rendered on a web page.

XML files can be generated manually. It is rather easy to write a simple XML file. What is simpler than to write the content of a town between two tags:

```
<town> what belongs to the town </town>
```

A more elaborate example is:

```
<village name=”Asten”>
    <place name=”Market”>
        <artifact> kiosk </artifact>
    </place>
    <street name=”Church alley” quarter=”12”>
        <house number=”2”> apartment </house>
        <house number=”4”> cottage </house>
        <shop number=”6”> grocery </shop>
        <parkingPlace name=”Corner parking"/>
    </street>
</village>
```

The example shows that the content that belongs to a tag can be empty and that tags can contain attributes. What attributes are possible, and their types and properties are specified in the DTD or schema files. Usually an XML file will not be generated or interpreted manually. Many tools can already generate and interpret XML files.

XML shares many characteristics with databases. It makes use of data dictionaries. These are present in the form of DTD or schema files. The XML files can be queried for specific content by using the Extendable Query Language (XQL). XML files contain plain text in which mark-up specifies the structure of the content. The difference with databases is, that databases are better optimized for data that occur in large series of similar items.

Standards for a series of associated files are in preparation. Amongst them are namespaces, eXtensible Link Language files and eXtensible Pointer Language files. XPL

[5] DTD files are an inheritance of SGML. Schema files can specify the hierarchy of an XML file in much more detail. Another advantage is that schema files are themselves XML files.

is used to refer to other XML documents and XLL is used to refer to sections within the same or another XML document.

The proposed namespace standard will make it possible that DTD and schema files can inherit each other's definitions. This means that it becomes possible to map the inheritance of the states of instances of object-oriented classes onto the corresponding XML state container. It means that there is a close correspondence between XML technology and object-oriented technology and an even simpler correspondence between XML technology and software component technology[6].

Status of current XML technology

Problems with current XML technology are:

1. Not all required standards are finalized yet. The XML standard was accepted in February 1998, but the standards for namespaces, the XSL files, the schema files and XLL, XPL and XQL are still in the works.
2. Current browsers do not yet support the full functionality of XML and its partner files. Some recent versions of browsers, like Internet Explorer 5.0 and Navigator 4.7 provide early implementations that will change later.
3. Tools supporting XML are emerging with a fast pace but they are not yet abundant. Only when all related standards are accepted will tools become reliable and stable.

Exchanging state information

One of the basic differences between hardware and software objects is that software objects have a state that can be initialized, replaced, archived and restored. A hardware object has a state that can be influenced, but it is not replaceable, archiveable or restorable. However, in some situations components containing part of the state of the hardware system can be exchanged. The state of a hardware device is instantiated during the build of the object. The state is a structured set of attributes. For software objects the state consists of properties and references. XML is a bag that can contain a structured set of items. These items are represented in textual format.

A DTD or schema file specifies the required structure of a corresponding class of XML files. One or more XSL files may describe how the content of a member of this class will be rendered on a web page. XPL and XLL statements can link the items stored in the XML bag to more detailed information or they may help to refer to parts of the content that are included in a separate XML bag. XQL can be used to query XML files for specific content.

Thus, if the proper DTD or schema files are set, then XML files can be used as EXTERNAL CONTAINERS of the state of software objects. These software objects may be software components, local executables or even complete packages of distributed executables. For example the executables contained in such packages may together perform the immense job of data management in a large enterprise.

This makes XML the ideal vehicle for transferring, archiving, restoring and publishing states of systems and system components. It means, that in the future many repositories and interchange protocols that help in exchanging state information will use XML as the interchange format.

Distributed communication

Currently distributed communication between processes and over networks takes place by protocols that use complex marshalling in order to communicate object states between non-uniform environments. With XML it becomes possible to make this transfer much more transparent.

[6] More in-depth information on XML and its associated standards can be found on the web site of ISO workgroup W3C.

XML Metadata Interchange

Standardizing a common interchange format

XML files can be used as the medium for exchanging design elements between tools and between web-based repositories and tools. An important prerequisite is that for the target domain the corresponding DTD or schema files are made available to the participants in this interchange act. This is exactly what the Object Management Group pursues with its XML Metadata Interchange (XMI) standard. XMI treats models, meta-models and meta-meta-models. It uses XML for its exchange format, the Meta Object Facility (MOF) for its meta-modeling technology and the Unified Modeling Language (UML) for its modeling technology. In CORBA, MOF is used for modeling Interface Definition Language (IDL) files. UML is commonly used as a (graphical) system design language.

In XMI, models, meta-models and meta-meta-models contain packages, classifiers, classes, attributes, operations, constraints and associations as modeling elements. The list of modeling elements is much longer, but the list given here plays a major role. The XMI standard uses DTD files to define the scripts that will be used for the exchange of modeling information between tools and between tools and repositories. It means that for each of the modeling elements appropriate XML tags are defined in the MOF-DTD and/or UML-DTD. The XMI standard is still in preparation[7].

System design capabilities enabled by XMI

Current situation

Currently system design is mostly done manually or using a set of unconnected tools. Use is made of handbooks and reference books rather than by using publicly accessible repositories, from which machine-readable design information can be retrieved. This results in a complicated, time consuming and rather risky procedure.

Integral toolkit

System generation is best supported by an integrated set of tools that covers requirement specification, design, configuration, documentation, reuse-promotion, product and component marketing and maintenance. XMI plays an important role in reuse promotion and in acquiring existing components or design elements, such as types and interfaces. It supports the exchange of information between the members of the toolkit and it supports the exchange of information between tools and repositories. The toolkit will enable the construction of skeletons of components, for which the design information is retrieved from one or more domain related repositories. Skeletons can be used to build running and testable prototypes of the target system. The creation of a skeleton is much simpler than the generation of a fully functional component. For that reason it is healthy when the toolkit enables the creation of skeletons of components that are not available locally and that cannot be retrieved from accessible repositories.

Potential scenarios

In order to show the capabilities of the technology that will be enabled by XMI some potential scenarios are described below.

System generation

In a system integration firm a group of marketing experts analyze the possibilities for a new product and generate a set of requirements for a family of future products. A group of system architects translate the requirements into a design by reusing, as much as is possible components, which already exist locally. The require interfaces and provide interfaces that were used in these components were all taken from, or published to publicly available repositories. So the chance is large that these components can be coupled to other components that are also specified on these repositories. The information

[7] More in-depth information on XMI, MOF and UML can be found on the web site of the Object Management Group.

contained on the repositories is categorized according to a set of relevant domains and sub-domains. This eases the query for suitable information.
The group of architects search a set of interesting repositories for components that may cover required functionality that is still missing in the locally available set of components. At the same time they search for interfaces that relate to this extra functionality. If the wanted components cannot be found, then the architects design new components using the interfaces found or when no interface fits, by carefully designing their own interfaces. When they design new interfaces, they keep future reuse, possibly by other interested parties, in mind. The newly designed components will be designed for reuse and easy maintenance. It means that these components will be carefully documented and thoroughly tested. Later the commercial department may decide to make these components available for sale. It means that the passive relational architecture of the components with a highlight on types and interfaces will be published using XMI technology on one or more repositories.
When all required components are available, the system can be configured and a final test will ensure the correctness of the design. The advantage of this scenario over twentieth century scenarios is that system integration is much faster, costs less and is less risky. In an early stage of the system design process it becomes clear that the pursued concept is feasible. Potential customers can be confronted with running early prototypes. This makes it possible to adapt the set of requirements to new insights at a moment that changes are not so expensive as in the final stages of the system creation process.

Independent component generation

A small firm or a department of a large firm with typical domain expertise operates in a niche of the system component market. It encapsulates its domain knowledge in hardware components and software components and sells them as hybrid components. It uses interfaces that are as much as is possible taken from a series of popular repositories. This guarantees that their products can be coupled to a large series of available components and increases the chance that system integrators will make use of their components. Their marketing department uses several repositories to publish the passive relational architecture of the produced components. The published information contains all information that XMI enabled tools need to construct skeletons of the components. The system integrators can use these skeletons to generate prototypes of their target system. The published dataset contains an URL to the site of the supplier of the component. If the system integrator decides to use the component then he may use this URL to order the hybrid component via e-commerce. The contract may take seconds to complete. After that the active software components may be downloaded and the hardware components will be sent by express mail.

Corollary

XMI, which relies on XML, will enable a busy open market of system components. These components can be hardware components, software components or hybrid components. A series of web-based repositories together with integrated toolkits will be used to promote reuse by providing facilities for storage, retrieval, publishing, merchandising and skeleton creation of components. System generation will be much faster, less costly and less risky. Experts that have a different, more domain related skill than the system integrators of the twentieth century, can perform this job with great success.

Small companies and departments of large companies that operate in a niche of the market can still survive and make profit by selling components that encapsulate and hide their expert knowledge.

By using the technology that is described in this paper, a community of small firms can cooperate in order to create systems that conquer products, which are currently supplied by firms that completely dominate the market.