

# Business Process Automation and the Tower of Babel

There is a semantic gap between the people who have problems needing to be solved, and those attempting to solve them. There have been many attempts to bridge this gap and over time various technologies have evolved that push formal problem definition higher up in the enterprise. They tend to result in layers with a different language for each and they also tend to provide increasingly higher levels of abstraction.

In the early days of data processing, only one language was required to implement an application. Since then there has been an explosion of languages and dialects, and today analysts and programmers need to know dozens or more languages for even a basic application. There has long been a solution to this tower of Babel problem – a technology that has the ability to handle multiple languages on multiple platforms to solve the Business Process Modeling, Management, and Engineering (BPMME) problem whilst requiring analysts to know only one, simple, language. Proponents call it “Development Before the Fact<sup>m</sup>” and it does solve many of the problems facing the IT profession today.

## Introduction

---

There is a burgeoning market in tools that (separately) address Business Process Modeling and Management. Some use a proprietary modeling language, others use UML. None of them address Business Process Engineering, largely because of the intractability of making measurements of any kind on graphical elements that in many cases have no semantic content. Since UML dominates the modeling market today, this paper focuses on comparing 001 with UML based strategies for Business Process Automation (BPA) and Enterprise Application Integration (EAI), including OMG's "Model Driven Architecture", "UML Profile and Interchange Models for Enterprise Application Integration (EAI) Specification" and other related UML technologies and the UML-2 Superstructure. 001 goes a long way towards solving the Tower of Babel problem that plagues the industry today. For more information about 001 visit the Hamilton Technologies Web site at <http://www.world.std.com/~hti/>, and for more information about UML, visit the OMG website at <http://www.omg.org/>.

## The platform issue

---

In a recent interview conducted by ACM Queue Magazine<sup>1</sup>, Adam Bosworth (ex General Manager at Microsoft in charge of WebData/XML strategy, now SVP Advanced development at BEA) said:

".NET Philosophy - any language, one platform"

"Java Philosophy - one language, any platform"

"... languages today aren't used to thinking about their own data structures as queriable objects... will result in a major language extension..."

Following this is a discussion about whole new languages. Aspect Oriented programming is one such.

<sup>1</sup> <http://www.acm.org/cacm/1001/1001toc.html> CACM, Vol. 44, No. 10 October, 2001

However, it just adds more complexity to an already impossibly complex cognitive load. What is needed is a whole new platform that can manage multiple languages on multiple platforms, using one meta-language, preferably visual, to do so, and if it supports queryable objects, (as does 001), so much the better. This was the premise of UML, but it has arguably failed to deliver.

## The Problems

---

### Too many languages

Consider a contemporary multi tier architecture application using Web Services. How many different languages must a team using (say) RUP have to learn fluently to actually construct this system? A table (in no particular order) helps illustrate the problem.

Management	Requirements	Modeling	Programming	Protocols	Testing
MS Project	MSWord	Class Diagrams	Java	HTTP	IEEE 1175
CVS, etc.	MSExcel	Package Diagrams	Swing	SOAP	X/Winrunner
Synergy	PowerPoint	Object Diagrams	C#, VB, etc.	UDDI	Test Director
Change Synergy	Doors	Use Case Diagrams	csh/sh etc	JDBC	Defect Tracker
CMMI, SW-SMM, SE_CMM, etc.	Requisite Pro	Sequence Charts	Perl, Python, etc..	LDAP	
RM-ODP	Use Cases	Collaboration diagrams	Awk	XML	
Collaboration	Story Boards	State Charts	HTML	DNS	
Time tracker	Flow Charts	Activity Diagrams	make etc.	WSDL	
IEEE/EIA/ISO/IEC 12207	DFDs	Component Diagrams	ASN.1	JSP/ASP	
ISO 15288	Configuration	Interaction Diagrams	SQL	ODBC	
ISO 15504, etc.	Netscape Magnus and other confs	Deployment Diagrams	DB Schema		
PSP/TSP	Apache httpd	ROOM Charts	CORBA IDL		<b>Editors</b>
BSR/ISO/IEC 11770	Servlet Descriptors	SDL <sup>2</sup>	Deployment descriptors		vi, emacs, notepad
ISO 9000 series	Solaris	Message Sequence Charts	ebXML etc.		gdb, other debuggers
	Veritas	UML tool (Rose, StP, etc).	Solaris, Unix, Linux, etc.		Front Page, Star Office, etc.
	Linux	MOF	W2K etc.		Visual Age, etc.
			Oracle etc.		

**Figure 1: A partial list of languages that one might have to use in a typical project**

Undoubtedly there's more than the 80 or so listed here, especially if legacy systems (IBM mainframe, Microsoft client/server, etc.) are involved. XML has caused a proliferation of new languages, often producing gigantic documents containing hundreds, if not thousands, of statements. No one can be fluent in all these languages, let alone the 14 or more that UML comes with. See also the "Quagmire" at <http://www.software.org/quagmire/frampapr/frampapr.html>.

<sup>2</sup> Software Description Language

## Traceability

Worse, there's no traceability, even within the families (say management, or modeling). Products like Telelogic's Tau, Aonix' StP, and IBM's Rational Real Time (used to be ObjecTime) attempt to provide some traceability but, in practice, with unacceptably high overhead for any system of significant size. eXtreme Programming (XP) techniques address the testing issue adequately, but it is not clear that XP scales well.

## Engineering

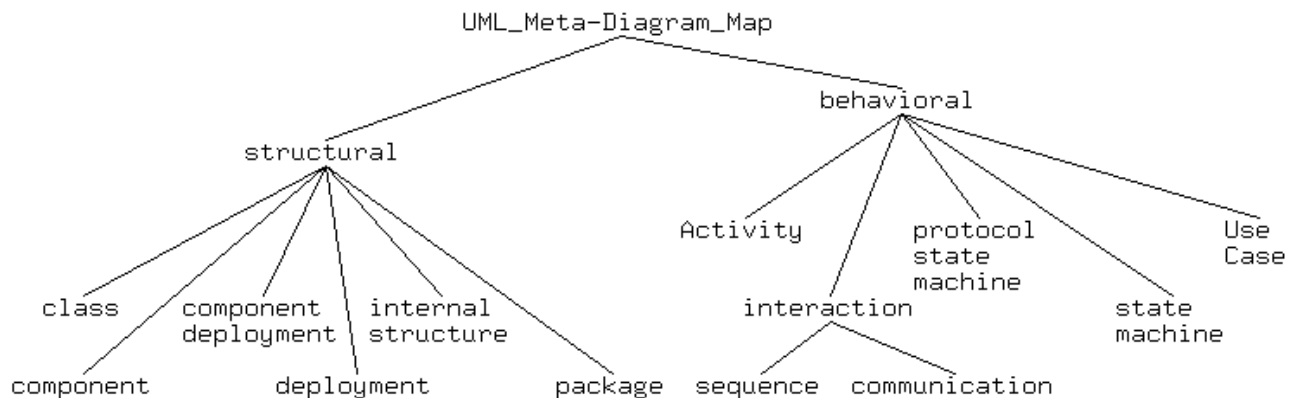
It is almost impossible to instrument UML drawings, although in principle StP allows one to do so, albeit again with a very high overhead. How many projects can truly state that the UML artifacts remaining at the end of the implementation phase bear any relevance to the implemented system? The net effect is that no practical continuous improvement process can be put in place for the most critical part of the whole development lifecycle. Errors at the topmost level can have huge impacts lower down where they can be enormously expensive to remedy.

## Solutions

---

### Classes of Diagrams vs. one simple modeling system based on tree like maps.

No one language or tool can solve the problem completely. However, reducing the number of languages can have some very significant benefits, as we'll see. Although in some sense XML has exacerbated the problem by creating a plethora of new languages, it has a unifying effect in that the same meta-language (XML itself) is used for all of them. Contrast this especially with UML where there is no consistency from one language to the next. UML-2 Superstructures has 16 different diagrams, including a diagram of diagrams; this meta-diagram (*map*) can be drawn simply in 001 as follows:



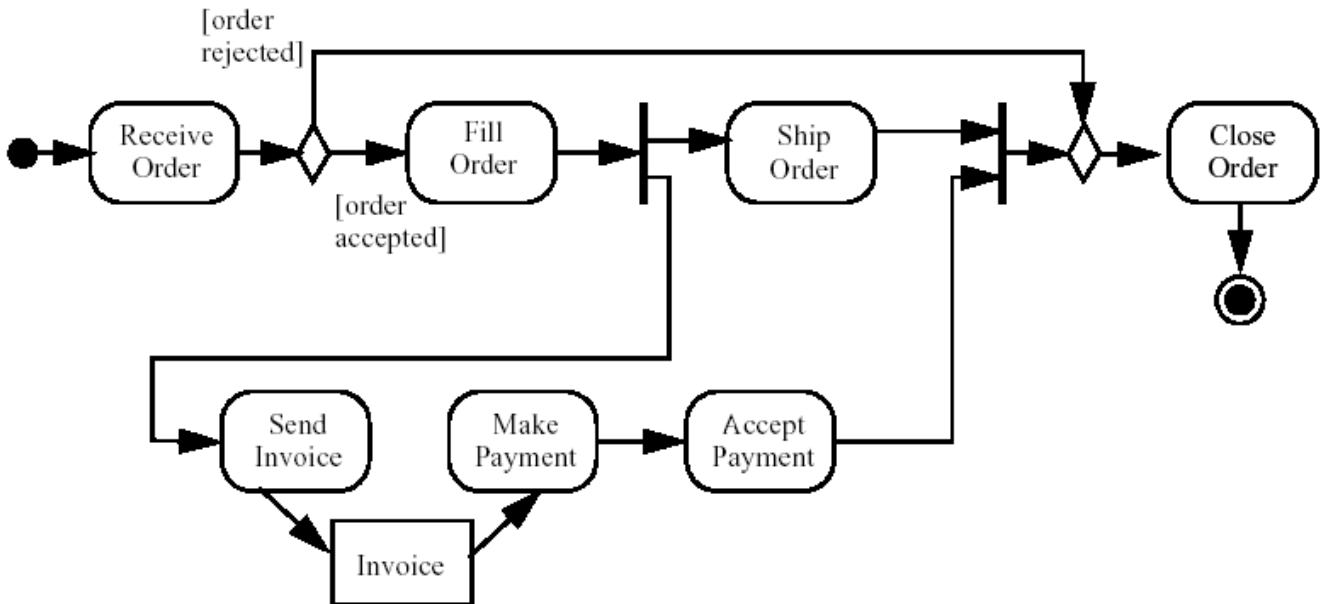
**Figure 2: Drawing of UML Diagram types**

001, it should be emphasized, only has one kind of drawing – a special kind of a tree called a *map*. Since we are interested in Business Process Modeling we'll concentrate for now on the behavioral drawings, but

001 is extremely adept at structural modeling as well and for completion, an 001 version of a Class Diagram is shown towards the end of this paper

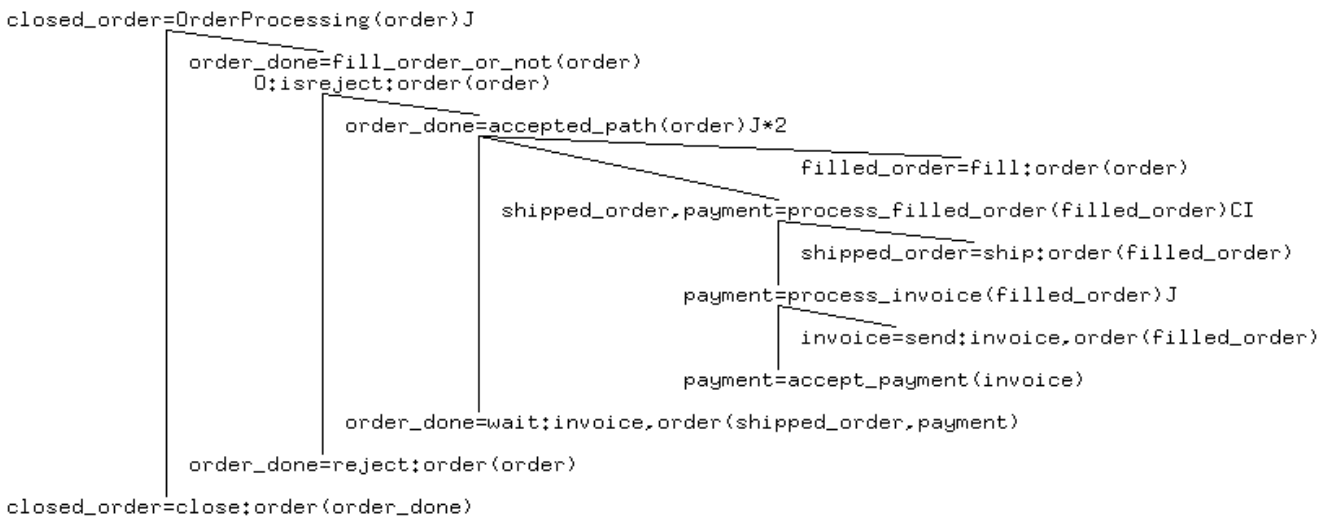
**Activity Diagram**

Consider a sample Activity Diagram describing the process of completing an order



**Figure 3: UML Activity Diagram from UML2 Superstructures**

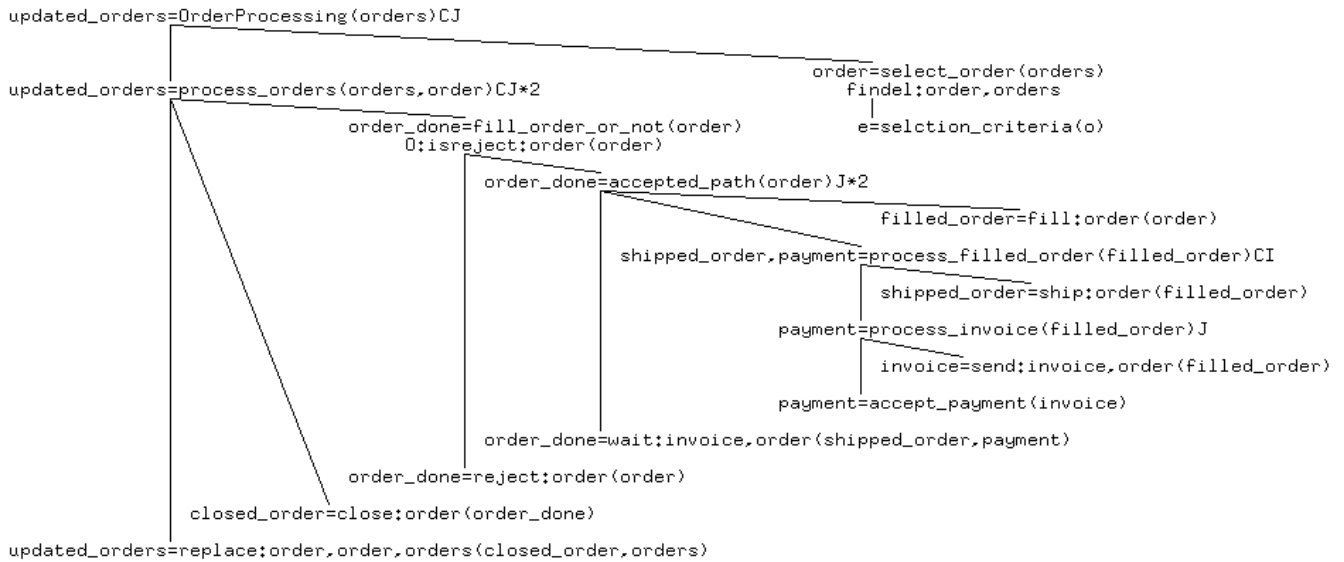
and a similar order process modeled in 001 (inputs on the right, outputs on the left, read top to bottom):



**Figure 4: 001 Initial Order Processing Example**

The parallelism implicit between shipping and invoicing in the UML drawing is explicit in the 001 model, and is actually identified automatically (nodes labeled with an “I” are parallel; “J” expresses dependency, here a sequential operation, and “O” represents a choice). Note also that the UML model explicitly mentions an object (an invoice) just once, but in reality there is an order object that oddly enough isn’t

shown on the diagram. The 001 model is also complete (verifiably so). One major difficulty with this UML model is that it ignores what it means to “receive” an order. It isn't really obvious what it means to close one either. Presumably there is some outside process that selects orders that the company has received and passes them into this set of processes one at a time, and that is what the 001 model makes clear. The close:order node on the 001 model makes no more sense than the one on the Activity diagram, and doesn't belong there either, since closing an order must affect some undefined entity (like a database or a filing cabinet) that is missing. This becomes obvious on the 001 model because one is forced into nonsense like having “order\_done” as well as “closed\_order”, which are clearly the same things. An improved version that makes all of this explicit would look something like this:

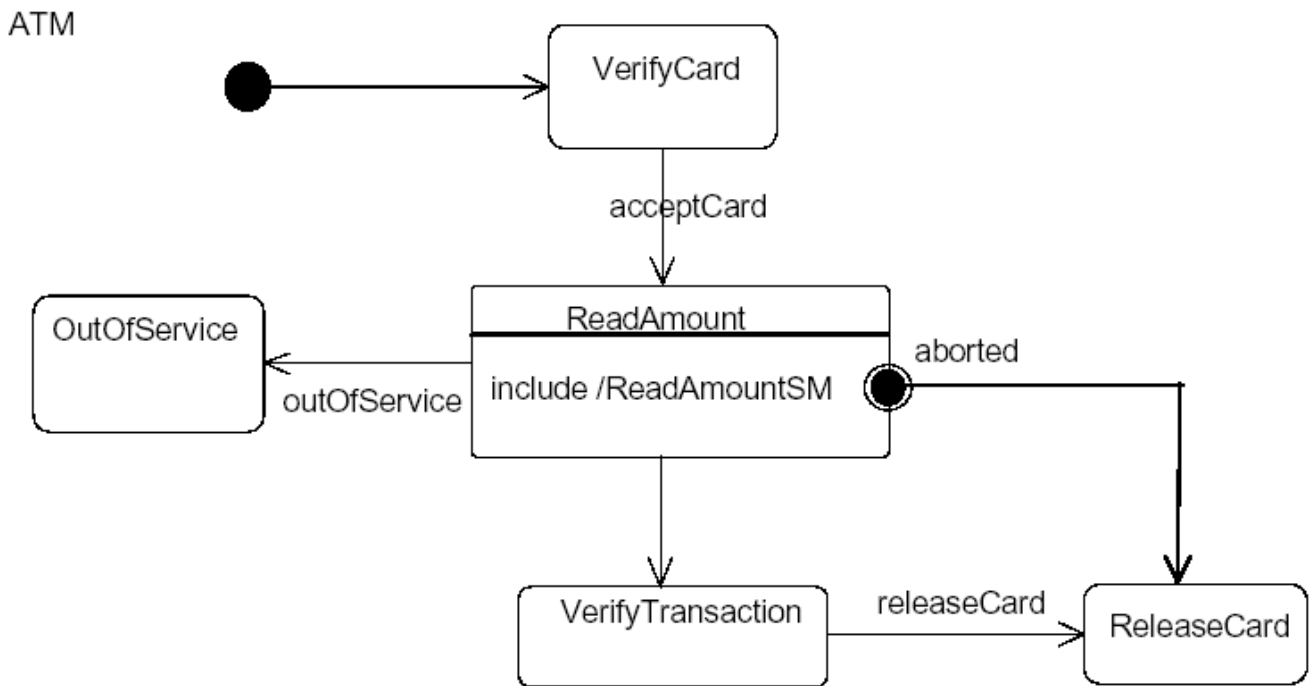


**Figure 5: Improved 001 Order Processing Example**

This version makes quite clear what the limitations and the applicability of the model are. This is a verifiably correct process model that can be implemented.

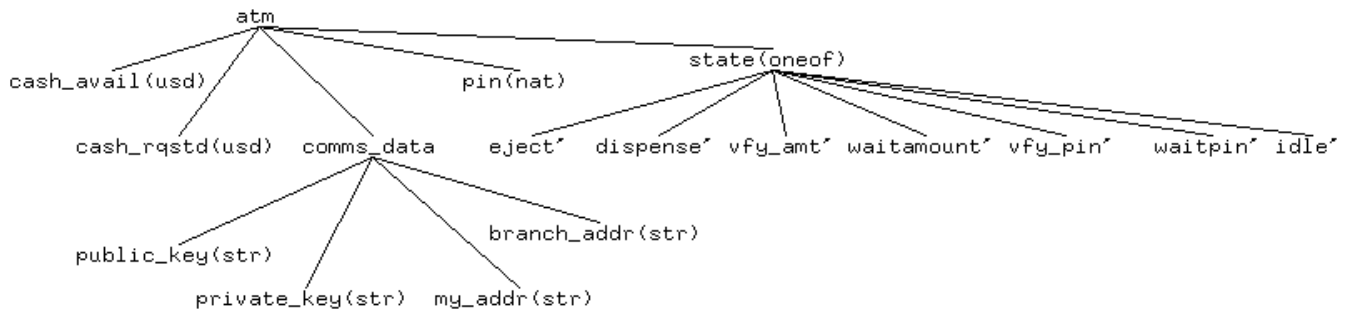
## State Diagram

For another example, consider a UML State Machine Diagram

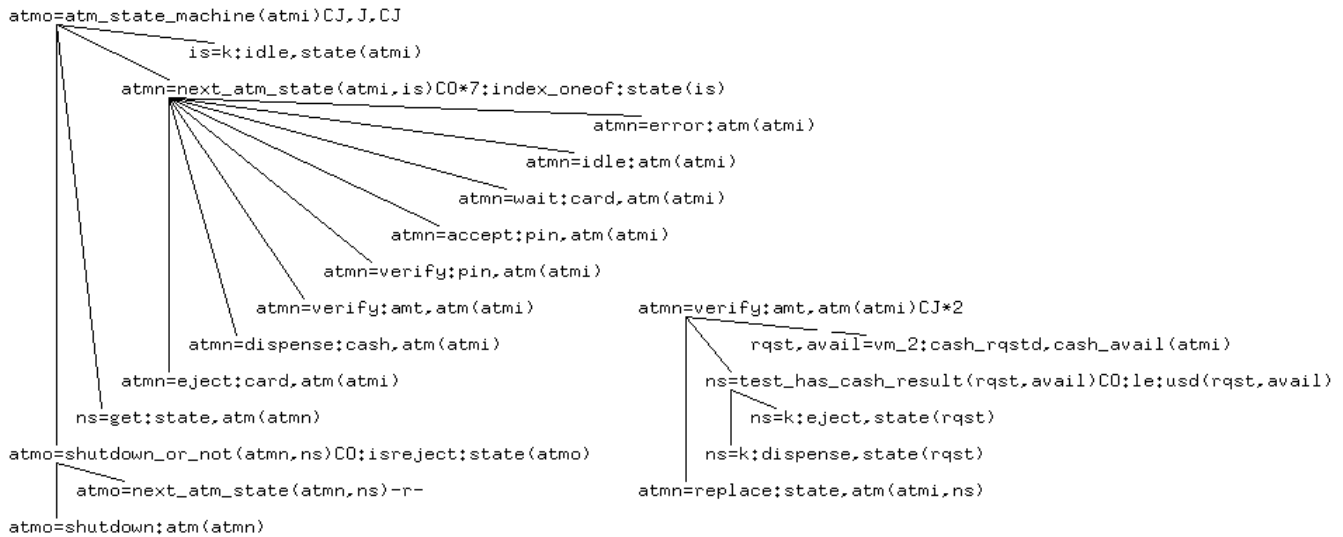


**Figure 6: UML2 State Machine diagram from UML2 Superstructures**

This particular UML model is so lacking in detail that it is hard to remodel accurately. It implies that if the ATM is out of service, the card is kept (since it isn't released) but my real world experience is that an out-of-service ATM won't accept the card to begin with. For completeness, here is the start of an entire ATM process state driven system model in 001 starting with the 001 equivalent of a Class Diagram.



**Figure 7: TMap for an ATM State Machine**



**Figure 8: FMap for an ATM State Machine**

One state action (verify amount) is shown completed (verify:amt,atm) and clearly shows the two possible new states resulting from a test of the available cash. The other states require similar decomposition.

### **Use Cases**

001 has the ability to automatically convert “Shall” statement style requirements into FMaps. It can also produce “Shall” style requirements of considerable detail from them (conforming to MIL STD 2167A) that read remarkably well for machine generated documentation. However, the challenge is to make some sense out of Use Cases and connect them to the rest of the modeling system, something that UML does not seem to manage well.

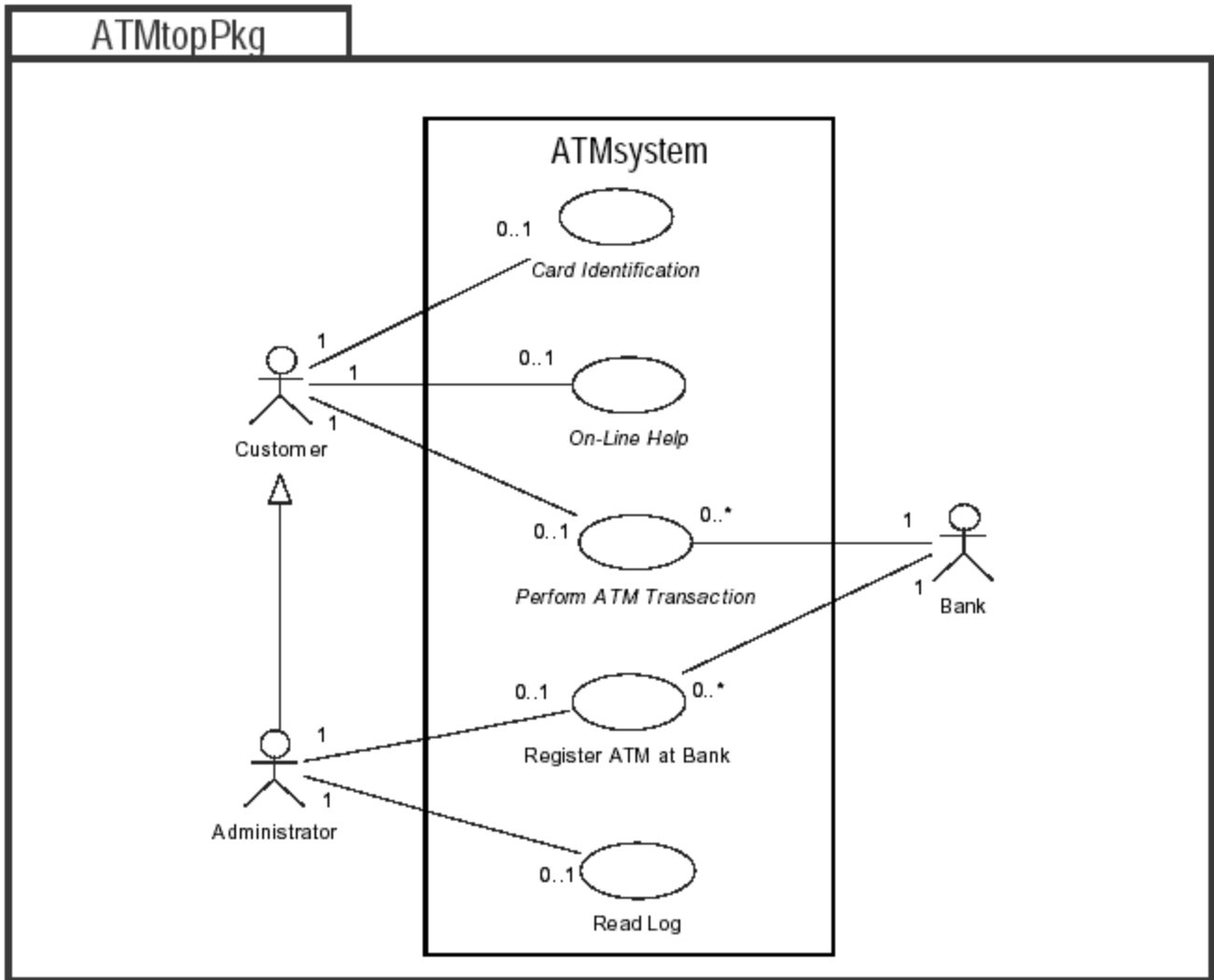


Figure 9 UML Use Case example from UML2 Superstructures

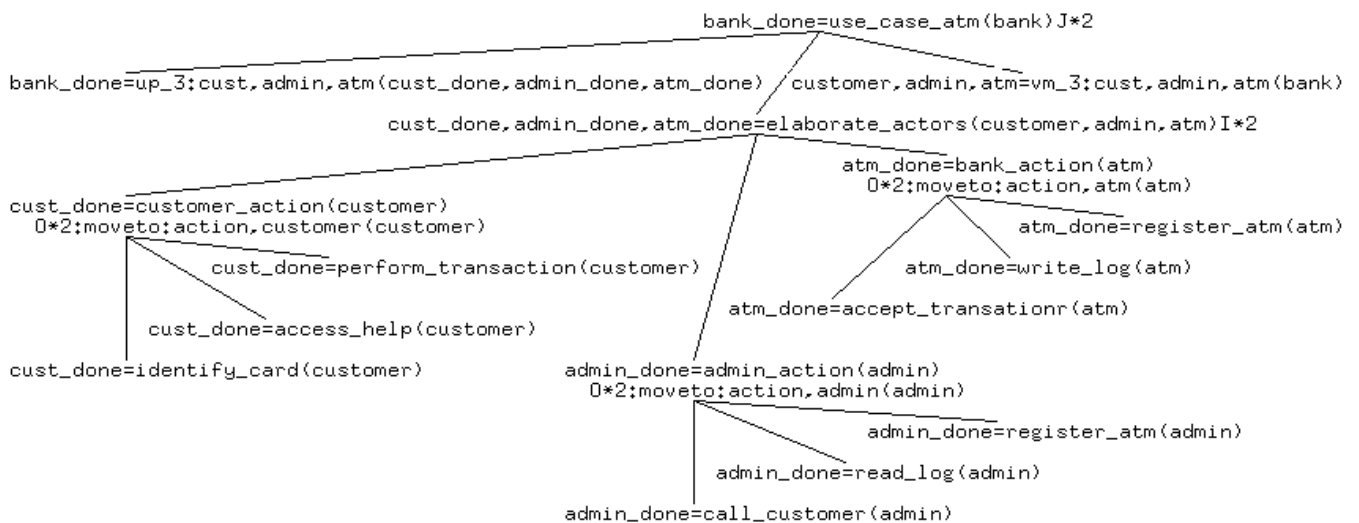
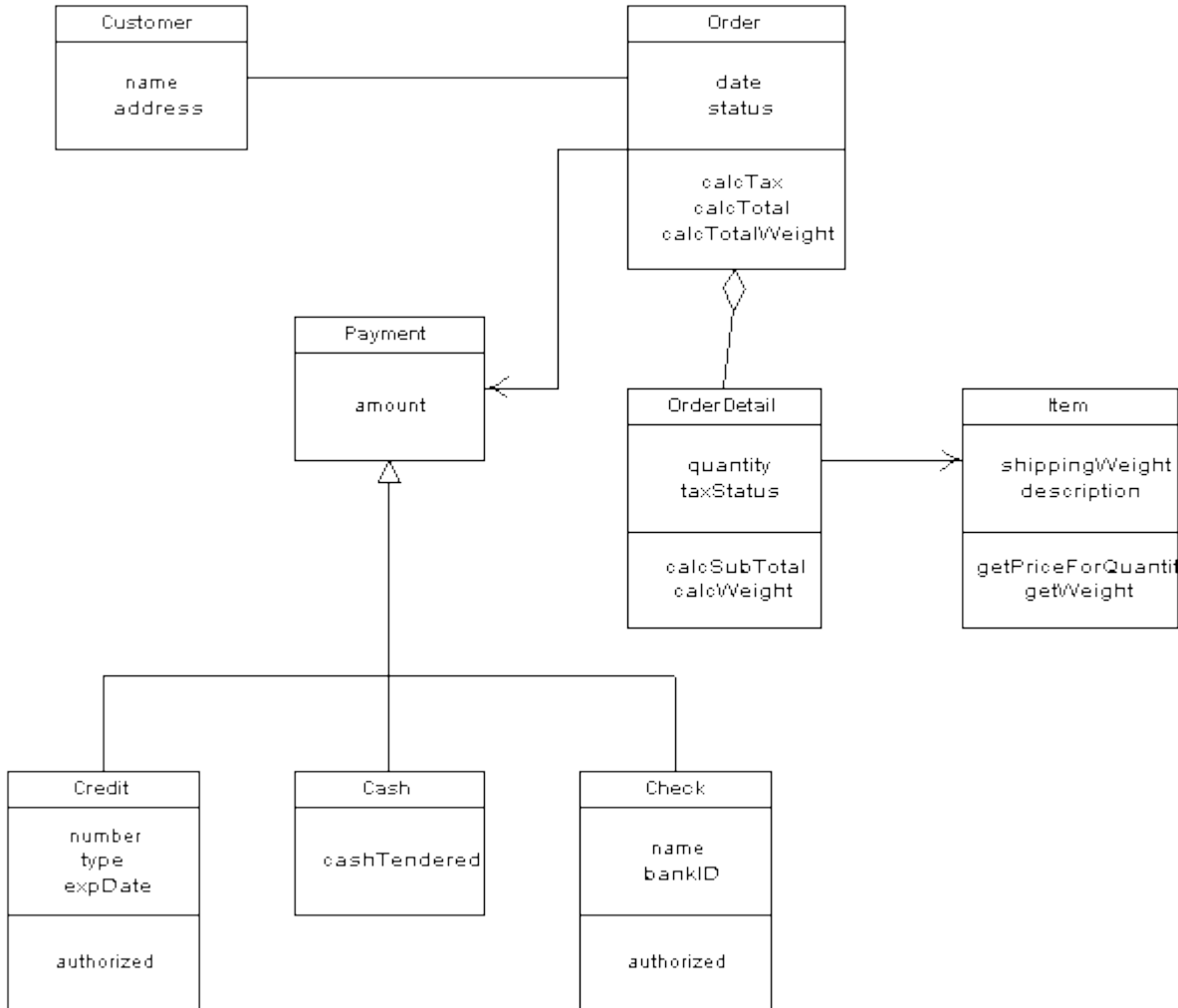


Figure 10: 001 Use Case Example

Similar conversions can be done with all the other UML drawings. It doesn't take long before you begin to realize that these diagrams are really just different representations of the same thing without really adding much comprehension, and that you really don't need 16 different kinds.

**Structural Diagrams**

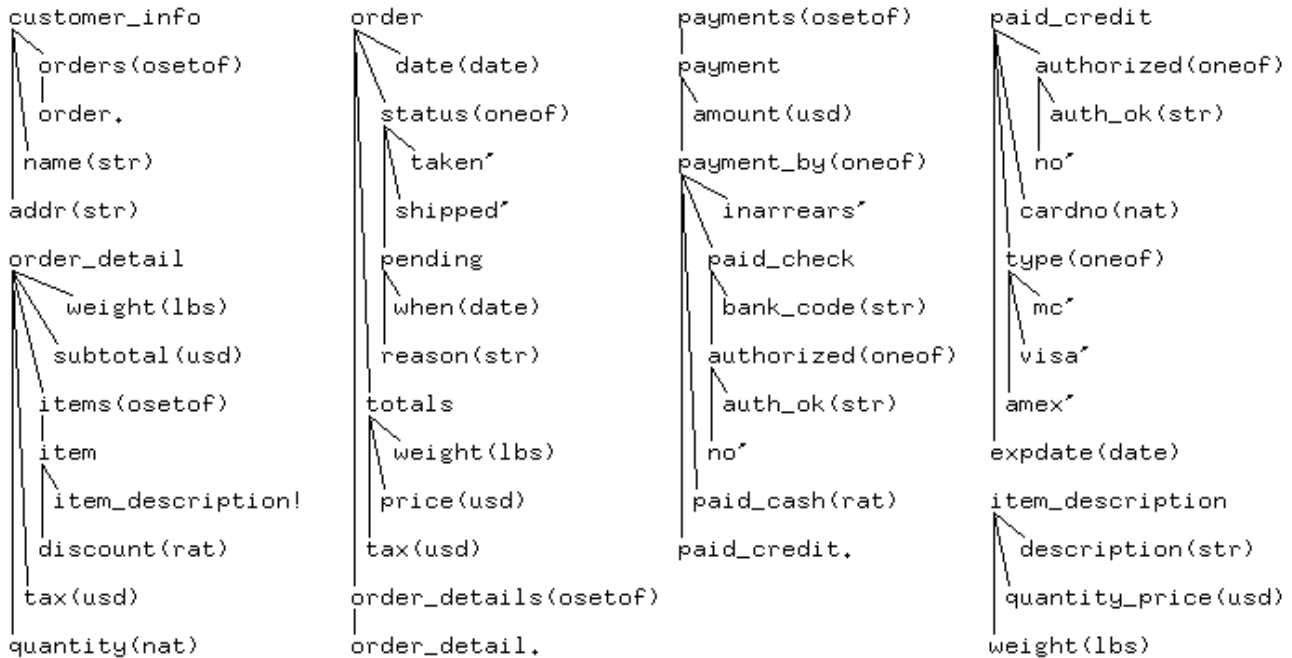
Finally, to illustrate the power of 001 of doing structural diagrams, consider this Class Diagram (similar to one in Borland's (now TogetherSoft) "Practical UML, A Hands-On Introduction for Developers").



**Figure 11: Retail Customer Order Class Diagram**

When the order is fulfilled, the Item attributes will apparently disappear. In a real system Items would be in a separate class as part of the Inventory system, and the 001 version of this clearly shows this by making the item a reference to an element of an inventory. UML does not apparently have a clear way of making this distinction. Note how the 001 high level model clearly drives the implementation, where the UML drawing does not. Note also that there's no need in 001 to enumerate the operations because 001 automatically generates every possible operation on an abstract type and actually creates help on operations on types for every type. This reduces the clutter on the high level models dramatically. 001 has sets (OsetOf) that can have zero or more identical elements, Tuples (TupleOf) that have a fixed number of

different elements, OneOfs that can take on one of several possible values, and TreeOf (not used here). All 001 objects are instances of abstract types in a navigable tree like *map* (Type Map or TMap) with primitive types at the leaf nodes. These primitives can be user defined – in this example, currency (usd) date (date) and weight (lbs) are user-defined primitive types. OsetOf is equivalent to a parallel operation on an FMap, TupleOf is analogous to a sequential operation (Join), and OneOf compares to an OR (switch). As a result there are merely 3 structure types (from which all others can be derived, such as those used in this article) that make up all 001 *maps*.



**Figure 12: 001 TMap for the retail customer example set of payments**

## Application to Business Process Automation

---

### Code Generation

001 can be used to model all the things that the 16 UML2 diagram types (as defined in more than 700 pages!), using one drawing style and 3 structure types that can be documented in a couple of pages. Importantly, the models can all be refined in a top-down fashion until the entire application has been modeled. Since 001 recognizes that there are only a limited set of operation on primitive types (add, subtract, etc.) and a finite set of operations on abstract types (creation, insertion, navigation, etc), it can generate production quality code from the models. Since it is language agnostic, it can generate code in any language, not limited to C, Java, Cobol, Fortran, etc., even English, but also shell languages and, most importantly, it can import and export XML schemas and easily convert instantiated objects into XML.

## **Process Management**

This means it can be used to model all levels of an enterprise, from the high level process view (say, the customer provisioning process), generating Web Services definitions, all the way down to the detailed specification and test of an individual component, generating C, csh, or whatever is needed. The tool provides everything needed to achieve full enterprise process management from XSLT to the humblest line of code, with full traceability of model to code and test.

## **Process Engineering**

001 has at its top level, a *map* of projects. A project is a *map* of libraries, and a library, a *map* of definitions. Since everything is done using the same language (with an underlying text form 001AXES<sup>m</sup>), it is very easy to keep statistics about projects, libraries, and definitions and how they changed. 001 offers extraordinary levels of reuse at the pattern level, and the combination of these features makes process engineering a reality. 250 lines of 001AXES a day are the same lines whether they are modeling a spacecraft, a supply chain, or the details of a backup process. Process Engineering is almost impossible to do with today's UML technology.

## **Structural Modeling**

There are many diagrams in UML2 (and more than 700 pages of documentation). .001 Type Maps, Road Maps, Function Maps, etc. can replace all of them using a unified and consistent user interaction, and using the same integrated visual *map* structures. The models can be used to generate code (including HTML, any XML, scripts, compilable code, and English documentation) to automate the process being modeled.

## **Process Modeling**

The 001Xecutor<sup>m</sup> is a mechanism for instrumenting the models and doing what-if measurements on them – add a resource here, change a cost there, etc., -- to do true cost and resource based modeling. It does it by associating super-objects with each object of interest (one step beyond the flat attributes associated with Java Beans), executing the model and reporting statistics as it runs. In many ways this is the inverse of the structural model and is complementary to it.

## **Testing**

The greatest failure of UML (other than the attempts by Aonix to instrument Use Cases in StP) is to add testability to the models. 001 has an XP-like ability to run models at any time during the elaboration cycle, in essence providing a complete workable system at all times. It does so by providing prompts for primitive types and a universal object editor for abstract types, so that any unimplemented feature can be emulated. A complete dump of all instantiated objects can be made at any time, thus facilitating regression testing.

## Conclusion

---

XML has great promise to unite the Data Processing world. However, at this time, there is a plethora of tools and interfaces being applied to every different facet of an organization. UML 2 is a gigantic catchall of dozens of different modeling languages with no consistency or traceability between each one. The language du jour approach to standardization is probably not useful in the long term. Historically each evolutionary step in programming has involved a greater level of abstraction, and OO1 is the ultimate in abstractive modeling tools because it is completely abstract. This may have held back its adoption, although this author has used it and it's predecessor very successfully at a number of large and small organizations. The time has come!

## About the Writer

---

Frank Middleton is president of Apogee Communications Technologies, Inc. (<http://www.apogeect.com>), providing consulting services in the New York Metro area in communications technologies - security, networking, systems architecture, design, implementation, certification, training and deployment. He has more than 20 years of experience in networking, security, infrastructure and applications architecture, design and implementation, is a long time member of IEEE and the ACM, and has a Masters in Computer Science from the Courant Institute of Mathematics, New York University. Presently Apogee is developing a meta-model of the business process modeling process, using the OO1 technology, with the goal of developing a tool that can help automate the process of automating business processes which is presently very labor intensive. He can be reached at 973-543-9324 or [f.middleton@apogeect.com](mailto:f.middleton@apogeect.com).